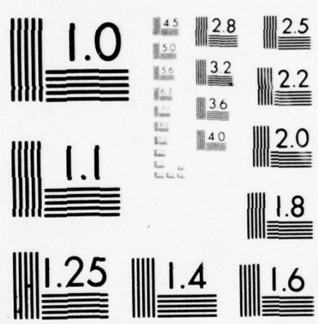
AD-A049 474   TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CALIF   F/G 9/2
SEMANOL (76) SPECIFICATION OF JOVIAL (J3). VOLUME III.(U)
NOV 77   F C BELZ, I M GREEN   F30602-76-C-0238

UNCLASSIFIED   RADC-TR-77-365-VOL-3   NL

1 OF 4
AD
A049474



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

RADC-TR-77-365, Vol III (of four)
Final Technical Report
November 1977

SEMANOL(76) SPECIFICATION OF JOVIAL(J3)

Frank C. Belz
Ira M. Green

TRW Defense and Space Systems Group

A049475

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-77-365, Vol III (of four) has been reviewed and approved for publication.

APPROVED: John M. Ives

JOHN M. IVES, Captain, USAF
Project Engineer

APPROVED: Alan R Barnum

ALAN R. BARNUM, Assistant Chief
Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

(19) TR-77-365-VOL-3

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: (18) RADC-TR-77-365, Vol III (of four)

2. GOVT ACCESSION NO.:

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): (6) SEMANOL(76) SPECIFICATION OF JOVIAL(J3). Volume III.

5. TYPE OF REPORT & PERIOD COVERED: (9) Final Technical Report.

6. PERFORMING ORG. REPORT NUMBER: N/A

7. AUTHOR(s): (10) Frank C. Belz, Ira M. Green

8. CONTRACT OR GRANT NUMBER(s): (15) F30602-76-C-0238

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
TRW Defense and Space Systems Group
One Space Park
Redondo Beach CA 90278

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
(16) P.E. 63728F
J.O. 55500840
(17) 08

11. CONTROLLING OFFICE NAME AND ADDRESS:
Rome Air Development Center (ISIS)
Griffiss AFB NY 13441

12. REPORT DATE: (11) Nov[redacted]77

13. NUMBER OF PAGES: 367 (12) 372 p.

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report):
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
Same

18. SUPPLEMENTARY NOTES:
RADC Project Engineer: Captain John M. Ives (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):
SEMANOL, SEMANOL(76), JOVIAL, JOVIAL(J3), semantics, syntax, language control, interpreter, metalanguage

20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

This report contains a formal specification of the JOVIAL(J3) programming language. The formal specification is written in TRW's SEMANOL(76) metalanguage, and so is a specification given in operational (i.e., interpretive) terms. The specification is generally complete and extremely detailed. This level of detail is unavoidable if JOVIAL(J3) semantics are to be comprehensively treated, as was done here. It also results from producing a specification that can actually produce operational results. However, by having an operational form of specification, it was possible to test the JOVIAL(J3) specification through use of the

next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409 637

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

SEMANOL(76) interpreter program. This testing insured the syntactic correctness of the entire metaprogram, and that much of the semantics yielded correct results for inputs of varying complexity.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

```
#DECLARE-GLOBAL:

     current-executable-unit,
     jovial-system,
     ncf-error-is-discovered,
     transformed-token-seq,
     unscanned-token-seq
     #.
```

```
#CONTEXT-FREE-SYNTAX:

"***<jovial-j3-system>***"

#DF jovial-j3-system

     => <gap> <jovial-j3-program> <gap> <optional-library>
        <optional-compools> <defaults> #.

#DF jovial-j3-program

     => <optional-control-input> <program> #.

#DF optional-control-input

     => <#NIL> #U <implementation-control-input> <gap> #.

#DF implementation-control-input

     => <'compools'> <gap> <':'> <gap> <compool-name-list>
        <gap> <'$'> #.

#DF compool-name-list

     => <selected-compool-name> <% <<gap>
        <selected-compool-name>>> #.

#DF selected-compool-name

     => <name> #.

     "***<optional-library>***"

#DF optional-library

     => <#NIL>
     => <'library'> <gap> <':'> <gap>
        <%1<<implementation-library-procedure> <gap>>> #.
```

```
#DF implementation-library-procedure

      => <optional-control-input> <<close-subprogram> #U
         <procedure-subprogram>> #.

#DF close-subprogram

      => <close-keyword> <gap> <close-name> <gap> <'$'> <gap>
         <close-subprogram-body> #.

#DF close-subprogram-body

      => <close-subprogram-start> <gap> <inner-close-body>
         <gap> <close-subprogram-term> #.

#DF close-subprogram-start

      => <start-phrase> #.

#DF start-phrase

      => <'START'> <gap> <'$'> #.

#DF close-subprogram-term

      => <term-phrase> #.

#DF term-phrase

      => <'TERM'> <gap> <'$'> #.

#DF procedure-subprogram

      => <procedure-head> <gap> <procedure-subprogram-body>
         #.

#DF procedure-subprogram-body

      => <procedure-subprogram-start> <gap>
         <proc-statement-list> <gap>
```

```
     <procedure-subprogram-term> #.


#DF procedure-subprogram-start

     => <start-phrase> #.


#DF procedure-subprogram-term

     => <term-phrase> #.

     "***<optional-compools>***"


#DF optional-compools

     => <#NIL>
     => <% <<implementation-compool> <gap>>> #.


#DF implementation-compool

     => <compool-header> <gap> <compool-body> #.


#DF compool-header

     => <'COMPOOL'> <gap> <name> <gap> <'$'> #.


#DF compool-body

     => <'START'> <gap> <'$'> <gap> <%1 <<compool-element>
        <gap>> > <'TERM'> <gap> <'$'> #.


#DF compool-element

     => <define-directive>
     => <simple-item-declaration>
     => <array-declaration>
     => <table-declaration>
     => <file-declaration>
     => <common-declaration>
     => <subprogram-declaration> #.
```

```
#DF common-declaration

     => <'COMMON'> <gap> <optional-common-block-name> <'$'>
        <gap> <begin-keyword> <gap> <%1
        <<common-block-element> <gap>>> <end-keyword> #.


#DF optional-common-block-name

     => <#NIL> #U <common-block-name> <gap> #.


#DF common-block-name

     => <name> #.


#DF common-block-element

     => <simple-item-declaration>
     => <array-declaration>
     => <table-declaration>
     => <independent-overlay-declaration>
     => <file-declaration> #.


#DF subprogram-declaration

     => <proc-keyword> <gap> <procedure-name> <gap>
        <optional-formal-parameter-list> <'$'> <gap>
        <optional-template-declarations> #.


#DF optional-template-declarations

     => <#NIL>
     => <begin-keyword> <gap> <%1<<template-declaration>
        <gap>>> <end-keyword> #.


#DF template-declaration

     => <table-declaration> #U <array-declaration> #U
        <simple-item-declaration> #.


#DF defaults
```

```
     => <'default'> <gap> <'declarations'> <gap> <':'> <gap>
        <%1<<default-declaration> <gap>>> #.


#DF default-declaration

     => <mode-directive>
     => <procedure-declaration> #.

     "***<program>***"


#DF program

     => <start-statement> <gap> <statement-list> <gap>
        <term-statement> #.


#DF start-statement

     => <'START', '[']START'> <gap> <optional-origin> <gap>
        <'$'> #.


#DF term-statement

     => <'TERM', '[']TERM'> <gap>
        <optional-initial-statement-name> <gap> <'$'> #.


#DF optional-initial-statement-name

     => <#NIL> #U < <initial-statement-name> <gap> > #.


#DF initial-statement-name
     => <name> #.


#DF statement-list

     => <statement-list-element> <% < <gap>
        <statement-list-element> > > #.


#DF statement-list-element
```

```
     => <statement> #U <declaration> #U <directive> #.

     "***<declaration>***"


#DF declaration

     => <data-declaration> #U <process-declaration> #.


#DF data-declaration

     => <simple-item-declaration> #U <array-declaration> #U
        <table-declaration> #U
        <independent-overlay-declaration> #U
        <file-declaration> #.


"***<simple-item-declaration>***"


#DF simple-item-declaration

     => <item-keyword> <gap> <simple-variable> <gap>
        <item-description> <gap> <'$'>
     => <item-keyword> <gap> <simple-variable> <gap>
        <item-description> <gap> <'P'> <gap>
        <optionally-signed-constant> <gap> <'$'>
     => <item-keyword> <gap> <simple-variable> <gap>
        <optionally-signed-constant> <gap> <'$'> #.


#DF item-keyword

     => <'ITEM', '[']ITEM'> #.


#DF item-description

     => <floating-item-description>
     => <integer-item-description>
     => <fixed-item-description>
     => <hollerith-item-description>
     => <status-item-description>
     => <boolean-item-description>
     => <transmission-code-item-description> #.
```

#DF floating-item-description

=> <'F'>
=> <'F'> <gap> <'R'> #.

#DF integer-item-description

=> <integer-specifier> <optional-rounding-specifier>
<optional-integer-value-range>#.

#DF optional-rounding-specifier

=> <#NIL>
=> <gap> <'R'> #.

#DF optional-integer-value-range

=> <#NIL>
=> <gap> <integer-constant> <gap> <'...'> <gap>
<integer-constant> #.

#DF fixed-item-description

=> <fixed-specifier> <optional-rounding-specifier>
<optional-fixed-value-range> #.

#DF optional-fixed-value-range

=> <#NIL>
=> <gap> <range-constant> <gap> <'...'> <gap>
<range-constant> #.

#DF range-constant

=> <integer-constant> #U <fixed-constant> #.

#DF fixed-specifier

=> <'A'> <gap> <numeral> <gap> <'S', 'U'> <gap>
<optional-sign> <numeral> #.

```
=======================================================================

#DF boolean-item-description

     => <'B'> #.

#DF hollerith-item-description

     => <'H'> <gap> <numeral> #.

#DF transmission-code-item-description

     => <'T'> <gap> <numeral> #.

#DF status-item-description

     => <'S'> <gap> <optional-numeral> <% <status-constant>>
        #.

#DF optional-numeral

     => <#NIL>
     => <numeral> <gap> #.

#DF integer-specifier

     => <'I', 'A'> <gap> <numeral> <gap> <'S', 'U'> #.

"***<array-declaration>***"

#DF array-declaration

     => <'ARRAY', '[']ARRAY'> <gap> <simple-variable> <gap>
        <dimension-list> <gap> <item-description> <gap>
        <'$'> <gap> <cf-constant-list> #.

#DF dimension-list

     => <numeral>
     => <numeral> <gap> <dimension-list> #.
```

```
#DF bare-constant-list

     => <optionally-signed-constant>
     => <optionally-signed-constant> <%1 <#SPACE>>
        <bare-constant-list> #.


#DF cf-constant-list

     => <list-begin-bracket> <bare-constant-list>
        <list-end-bracket>
     => <list-begin-bracket> <%1 <cf-constant-list>>
        <list-end-bracket>
     => <#NIL> #.

     "***<table-declaration>***"

#DF table-declaration

     => <ordinary-table-declaration>
     => <defined-entry-table-declaration>
     => <like-table-declaration> #.


#DF ordinary-table-declaration

     => <table-keyword> <gap> <optional-table-name> <gap>
        <table-size-specification> <gap>
        <optional-basic-structure-specification> <gap> <'$'>
        <gap> <begin-keyword> <gap>
        <ordinary-entry-description> <gap> <end-keyword> #.


#DF defined-entry-table-declaration

     => <table-keyword> <gap> <optional-table-name> <gap>
        <table-size-specification> <gap>
        <optional-basic-structure-specification> <gap>
        <numeral> <gap> <'$'> <gap> <begin-keyword> <gap>
        <defined-entry-description> <gap> <end-keyword> #.


#DF like-table-declaration

     => <table-keyword> <gap> <simple-variable> <gap>
        <optional-table-size-specification> <gap>
```

```
        <optional-basic-structure-specification> <gap>
        <optional-packing-specification> <gap> <'L'> <gap>
        <'$'> #.


#DF table-keyword

     => <'TABLE', '[']TABLE'> #.


#DF begin-keyword

     => <'BEGIN', '[']BEGIN'> #.


#DF end-keyword

     => <'END', '[']END'> #.


#DF optional-table-name

     => <table-name> #U <#NIL> #.


#DF table-size-specification

     => <'V'> <gap> <numeral>
     => <'R'> <gap> <numeral> #.


#DF optional-table-size-specification

     => <table-size-specification> #U <#NIL> #.


#DF optional-basic-structure-specification

     => <'P','S',#NIL> #.


#DF optional-packing-specification

     => <'N','M','D',#NIL> #.


#DF ordinary-entry-description
```

```
     => <%1 <<ordinary-table-item-declaration> <gap> #U
        <subordinate-overlay-declaration> <gap> >> #.

"***<ordinary-table-item-declaration>***"

#DF ordinary-table-item-declaration

     => <item-keyword> <gap> <simple-variable> <gap>
        <item-description> <gap> <'$'> <gap>
        <optional-one-dimensional-constant-list> #.

#DF one-dimensional-constant-list

     => <list-begin-bracket> <bare-constant-list>
        <list-end-bracket> #.

#DF optional-one-dimensional-constant-list

     => <one-dimensional-constant-list> #U <#NIL> #.

     "***<subordinate-overlay-declaration>***"

#DF subordinate-overlay-declaration

     => <overlay-keyword> <gap> <overlay-specification>
        <gap> <'$'> #.

#DF overlay-keyword

     => <'OVERLAY', '[']OVERLAY'> #.

#DF overlay-specification

     => <data-sequence>
     => <overlay-specification> <gap> <'='> <gap>
        <data-sequence> #.

#DF data-sequence

     => <simple-variable>
```

```
     => <data-sequence> <gap> <','> <gap> <simple-variable>
        #.

#DF defined-entry-description

     => <%1 < <string-item-declaration> <gap> #U
        <defined-entry-item-declaration> <gap> >> #.

#DF string-item-declaration

     => <'STRING', '[']STRING'> <gap> <simple-variable>
        <gap> <item-description> <gap> <numeral> <gap>
        <numeral> <gap> <optional-packing-specification>
        <gap> <numeral> <gap> <numeral> <gap> <'$'> <gap>
        <optional-two-dimensional-constant-list> #.

#DF optional-two-dimensional-constant-list

     => <list-begin-bracket> <gap> <%1 <
        <one-dimensional-constant-list> <gap> > >
        <list-end-bracket>
     => <#NIL> #.

#DF defined-entry-item-declaration

     => <item-keyword> <gap> <simple-variable> <gap>
        <item-description> <gap> <numeral> <gap> <numeral>
        <gap> <optional-packing-specification> <gap> <'$'>
        <gap> <optional-one-dimensional-constant-list> #.

#DF file-declaration

     => <'FILE', '[']FILE'> <gap> <file-name> <gap>
        <file-structure-specification> <gap> <status-list>
        <gap> <device-name> <gap> <'$'> #.

#DF file-structure-specification

     => <'B','H'> <gap> <numeral> <gap> <'V','R'> <gap>
        <numeral> #.
```

```
#DF status-list

     => <%1 <<status-constant> <gap>>> #.

#DF device-name

     => <name> #.

#DF independent-overlay-declaration

     => <overlay-keyword> <gap> <overlay-specification>
        <gap> <'$'>
     => <overlay-keyword> <gap> <numeral> #U
        <octal-constant> <gap> <'='> <gap>
        <overlay-specification> <gap> <'$'> #.

#DF process-declaration

     => <program-declaration> #U <close-declaration> #U
        <procedure-declaration> #U <switch-declaration> #.

#DF program-declaration

     => <'[']PROGRAM'> <gap> <program-name> <gap>
        <optional-origin> <gap> <'$'> #.

#DF program-name

     => <name> #.

#DF optional-origin

     => <numeral> #U <octal-constant> #U <#NIL> #.

#DF close-declaration

     => <close-keyword> <gap> <close-name> <gap> <'$'> <gap>
        <close-body> #.

#DF close-keyword
```

```
     => <'CLOSE', '[']CLOSE'> #.

#DF close-body

     => <begin-keyword> <gap> <inner-close-body> <gap>
        <close-end> #.

#DF inner-close-body

     => <statement-list> #.

#DF close-end

     => <end-keyword> #.

#DF close-name

     => <name> #.

#DF procedure-declaration

     => <procedure-head> <gap> <procedure-body> #.

#DF procedure-head

     => <proc-keyword> <gap> <procedure-name> <gap>
        <optional-formal-parameter-list> <'$'> <gap>
        <optional-decl-list> #.

#DF proc-keyword

     => <'PROC', '[']PROC'> #.

#DF procedure-name

     => <name> #.

#DF optional-formal-parameter-list
```

```
     => <#NIL>
     => <'('> <gap> <optional-input-parameter-list> <gap>
        <')'> <gap>
     => <'('> <gap> <optional-input-parameter-list> <gap>
        <'='> <gap> <output-parameter-list> <gap> <')'>
        <gap> #.


#DF optional-decl-list

     => <#NIL> #U <decl-list> #.


#DF decl-list

     => <decl-list-declaration> <% < <gap>
        <decl-list-declaration> > > #.


#DF decl-list-declaration

     => <directive> #U <data-declaration> #U
        <program-declaration> #.


#DF formal-input-parameter

     => <name>
     => <formal-input-close-parameter> <gap> <'.'> #.


#DF formal-input-close-parameter

     => <name> #.


#DF input-parameter-list

     => <formal-input-parameter>
     => <input-parameter-list> <gap> <','> <gap>
        <formal-input-parameter> #.


#DF optional-input-parameter-list

     => <#NIL>
     => <input-parameter-list> #.
```

```
#DF formal-output-parameter

     => <name>
     => <formal-output-destination-parameter> <gap> <'.'> #.

#DF formal-output-destination-parameter

     => <name> #.

#DF output-parameter-list

     => <formal-output-parameter>
     => <output-parameter-list> <gap> <'-'> <gap>
        <formal-output-parameter> #.

#DF procedure-body

     => <begin-bracket> <gap> <proc-statement-list> <gap>
        <procedure-end> #.

#DF proc-statement-list

     => <% <statement #U restricted-declaration #U
        directive>> #.

#DF restricted-declaration

     => <data-declaration> #U
        <restricted-process-declaration> #.

#DF restricted-process-declaration

     => <program-declaration> #U <close-declaration> #U
        <switch-declaration>#.

#DF procedure-end

     => <end-keyword> #.
```

```
#DF switch-declaration

     => <index-switch-declaration>
     => <item-switch-declaration>#.

#DF index-switch-declaration

     => <switch-keyword> <gap> <switch-name> <gap> <'='>
        <gap> <'('> <gap> <index-switch-list> <gap> <')'>
        <gap> <'$'> #.

#DF switch-keyword

     => <'SWITCH', '[']SWITCH'> #.

#DF index-switch-list

     => <optional-sequence-designator>
     => <optional-sequence-designator> <gap> <','> <gap>
        <index-switch-list> #.

#DF optional-sequence-designator

     => <<sequence-designator> #U <nil>> #.

#DF nil

     => <#NIL> #.

#DF switch-name

     => <name>#.

#DF item-switch-declaration

     => <switch-keyword> <gap> <switch-name> <gap> <'('>
        <gap> <simple-variable> <gap> <')'> <gap> <'='>
        <gap> <'('> <gap> <item-switch-list> <gap> <')'>
        <gap> <'$'> #.
```

==

#DF item-switch-list

=> <item-switch-case-expression>
=> <item-switch-case-expression> <gap> <','> <gap>
<item-switch-list> #.

#DF item-switch-case-expression

=> <general-constant> <gap> <'='> <gap>
<sequence-designator> #.

#DF general-constant

=> <numeric-constant> #U <string-constant> #U
<status-constant>#.

#DF statement

=> <optional-statement-name-list> <unnamed-statement>
#.

#DF unnamed-statement

=> <independent-statement>
=> <complex-statement>#.

#DF optional-statement-name-list

=> <% < <statement-label> <gap> > > #.

#DF statement-label

=> <statement-name> <gap> <'.'> #.

#DF statement-name

=> <name> #.

#DF independent-statement

```
     => <simple-statement> #U <compound-statement>#.


#DF simple-statement

     => <assignment-statement>
     => <exchange-statement>
     => <go-to-statement>
     => <test-statement>
     => <return-statement>
     => <stop-statement>
     => <procedure-call-statement>
     => <input-statement>
     => <output-statement>
     => <open-input-statement>
     => <open-output-statement>
     => <shut-input-statement>
     => <shut-output-statement> #.


#DF assignment-statement

     => <numeric-variable> <gap> <'='> <gap>
        <numeric-expression> <gap> <'$'>
     => <boolean-variable> <gap> <'='> <gap>
        <boolean-expression> <gap> <'$'>
     => <variable> <gap> <'='> <gap> <atomic-formula> <gap>
        <'$'> #.


#DF exchange-statement

     => <variable> <gap> <'=='> <gap> <variable> <gap> <'$'>
        #.


#DF go-to-statement

     => <'GOTO', '[']GOTO'> <gap> <sequence-designator>
        <gap> <'$'> #.


#DF sequence-designator

     => <destination-name>
     => <destination-name> <gap> <'($'> <gap>
        <destination-index> <gap> <'$)'> #.
```

```
#DF destination-name

     => <name> #.


#DF destination-index

     => <index-list> #.


#DF test-statement

     => <test-keyword> <gap> <'$'>
     => <test-keyword> <gap> <loop-variable> <gap> <'$'> #.


#DF test-keyword

     => <'TEST', '[']TEST'> #.


#DF return-statement

     => <'RETURN', '[']RETURN'> <gap> <'$'> #.


#DF stop-statement

     => <'STOP', '[']STOP'> <gap>
        <optional-initial-statement-name> <'$'> #.


#DF procedure-call-statement

     => <procedure-name> <gap> <'$'>
     => <procedure-name> <gap> <'('> <gap>
        <optional-actual-input-parameter-list> <gap> <')'>
     => <procedure-name> <gap> <'('> <gap>
        <optional-actual-input-parameter-list> <gap> <'='>
        <gap> <actual-output-parameter-list> <gap> <')'>
        <gap> <'$'> #.


#DF actual-output-parameter

     => <variable>
```

```
     => <actual-output-destination-parameter> <gap> <'.'> #.


#DF actual-output-destination-parameter

     => <name> #.


#DF actual-output-parameter-list

     => <actual-output-parameter>
     => <actual-output-parameter-list> <gap> <','> <gap>
        <actual-output-parameter> #.


#DF input-statement

     => <input-keyword> <gap> <file-name> <gap>
        <input-operand> <gap> <'$'> #.


#DF input-keyword

     => <'INPUT', '[']INPUT'> #.


#DF output-statement

     => <output-keyword> <gap> <file-name> <gap>
        <output-operand> <gap> <'$'> #.


#DF output-keyword

     => <'OUTPUT', '[']OUTPUT'> #.


#DF input-operand

     => <variable>
     => <name> <gap> <'($'> <gap> <index-list> <gap> <'...'>
        <gap> <index-list> <gap> <'$)'> #.


#DF output-operand

     => <constant> #U <string-constant>
     => <input-operand> #.
```

```
#DF open-input-statement

     => <open-keyword> <gap> <input-keyword> <gap>
        <file-name> <gap> <'$'>
     => <open-keyword> <gap> <input-keyword> <gap>
        <file-name> <gap> <input-operand> <gap> <'$'> #.


#DF open-output-statement

     => <open-keyword> <gap> <output-keyword> <gap>
        <file-name> <gap> <'$'>
     => <open-keyword> <gap> <output-keyword> <gap>
        <file-name> <gap> <output-operand> <gap> <'$'> #.


#DF open-keyword

     => <'OPEN', '[']OPEN'> #.


#DF shut-input-statement

     => <shut-keyword> <gap> <input-keyword> <gap>
        <file-name> <gap> <'$'>
     => <shut-keyword> <gap> <input-keyword> <gap>
        <file-name> <gap> <input-operand> <gap> <'$'> #.


#DF shut-output-statement

     => <shut-keyword> <gap> <output-keyword> <gap>
        <file-name> <gap> <'$'>
     => <shut-keyword> <gap> <output-keyword> <gap>
        <file-name> <gap> <output-operand> <gap> <'$'> #.


#DF shut-keyword

     => <'SHUT', '[']SHUT'> #.


#DF compound-statement

     => <begin-bracket> <gap> <statement-list> <gap>
        <end-bracket> #.
```

```
#DF begin-bracket

     => <begin-keyword> #.

#DF end-bracket

     => <end-keyword> #.

#DF complex-statement

     => <direct-statement>
     => <conditional-statement>
     => <alternative-statement>
     => <loop-statement>#.

#DF direct-statement

     => <'DIRECT', '[']DIRECT'> <gap> <direct-code> <gap>
        <'JOVIAL', '[']JOVIAL'> #.

#DF basic-direct-code

     => <implementation-dependent-direct-code>
     => <direct-assign>#.

#DF implementation-dependent-direct-code

     => <#SPACE> #.

#DF direct-assign

     => <assign-keyword> <gap> <'A'> <
        <optionally-signed-integer-constant> #U <#NIL> >
        <gap> <'='> <gap> <declared-variable> <gap> <'$'>
     => <assign-keyword> <gap> <declared-variable> <gap>
        <'='> <gap> <'A'> <
        <optionally-signed-integer-constant> #U <#NIL> >
        <gap> <'$'> #.
```

```
#DF assign-keyword

     => <'ASSIGN', '[']ASSIGN'> #.


#DF direct-code

     => <basic-direct-code>
     => <direct-code> <basic-direct-code> #.


#DF conditional-statement

     => <if-keyword> <gap> <boolean-formula> <gap> <'$'>
        <gap> <independent-statement>
        <unwritten-conditional-end> #.


#DF if-keyword

     => <'IF', '[']IF'> #.


#DF unwritten-conditional-end

     => <#NIL> #.


#DF alternative-statement

     => <alternative-list> <gap> <alternative-statement-end>
        #.


#DF alternative-statement-end

     => <end-keyword> #.


#DF alternative-list

     => <if-either-alternative> <%1 < <gap>
        <or-if-alternative> > > #.


#DF if-either-alternative

     => <'IFEITH', '[']IFEITH'> <gap> <boolean-formula>
```

```
        <gap> <'$'> <gap> <independent-statement>
        <unwritten-alternative-end> #.


#DF or-if-alternative

     => <optional-statement-name-list> <'ORIF', '[']ORIF'>
        <gap> <boolean-formula> <gap> <'$'> <gap>
        <independent-statement> <unwritten-alternative-end>
        #.


#DF unwritten-alternative-end

     => <#NIL> #.


#DF loop-statement

     => <complete-loop-header> < <independent-statement> #U
        <special-compound> >
     => <two-factor-loop-header> < <independent-statement>
        #U <special-compound> >
     => <one-factor-loop-header> <independent-statement> #.


#DF complete-loop-header

     => <optional-one-factor-for-clause-list>
        <complete-for-clause> <gap>
        <optional-one-and-two-factor-for-clause-list> #.


#DF two-factor-loop-header

     => <optional-one-factor-for-clause-list>
        <two-factor-for-clause> <gap>
        <optional-one-and-two-factor-for-clause-list> #.


#DF one-factor-loop-header

     => <optional-one-factor-for-clause-list>
        <one-factor-for-clause> <gap> #.


#DF optional-one-factor-for-clause-list
```

```
    => <% < <one-factor-for-clause> <gap> > > #.

#DF optional-one-and-two-factor-for-clause-list

    => <% < <<one-factor-for-clause> #U
       <two-factor-for-clause>> <gap> > > #.

#DF one-factor-for-clause

    => <for-keyword> <gap> <loop-variable> <gap> <'='>
       <gap> <initial-formula> <gap> <'$'> #.

#DF for-keyword

    => <'FOR', '[']FOR'> #.

#DF two-factor-for-clause

    => <for-keyword> <gap> <loop-variable> <gap> <'='>
       <gap> <initial-formula> <gap> <','> <gap>
       <increment-formula> <gap> <'$'> #.

#DF complete-for-clause

    => <for-keyword> <gap> <loop-variable> <gap> <'='>
       <gap> <initial-formula> <gap> <','> <gap>
       <increment-formula> <gap> <','> <gap>
       <termination-formula> <gap> <'$'>
    => <for-keyword> <gap> <loop-variable> <gap> <'='>
       <gap> <all-keyword> <gap> <'('> <gap> <tabular-name>
       <gap> <')'> <gap> <'$'> #.

#DF all-keyword

    => <'ALL', '[']ALL'> #.

#DF initial-formula

    => <numeric-formula> #.
```

```
#DF increment-formula

     => <numeric-formula> #.

#DF termination-formula

     => <numeric-formula> #.

#DF unnamed-special-compound

     => <begin-keyword> <gap> <statement-list> <gap>
        <if-keyword> <gap> <special-test-statement> <gap>
        <'$'> <gap> <end-keyword> #.

#DF special-test-statement

     => <boolean-formula> #.

#DF special-compound

     => <optional-statement-name-list>
        <unnamed-special-compound> #.

#DF directive

     => <define-directive>
     => <mode-directive> #.

#DF define-directive

     => <'DEFINE','[']DEFINE'> <token-gap> <defined-name>
        <restricted-token-gap> <double-prime> <definiens>
        <double-prime> <token-gap> <'$'> #.

#DF mode-directive

     => <mode-keyword> <gap> <item-description> <gap> <'$'>
     => <mode-keyword> <gap> <item-description> <gap> <'P'>
        <gap> <optionally-signed-constant> <gap> <'$'> #.
```

```
#DF mode-keyword

     => <'MODE', '[']MODE'> #.


#DF numeric-formula

     => <atomic-numeric-formula>
     => <numeric-expression>#.


#DF numeric-expression

     => <numeric-term>
     => <sum>
     => <difference> #.


#DF sum

     => <numeric-formula> <gap> <'+'> <gap> < <numeric-term>
        #U <atomic-numeric-formula> > #.


#DF difference

     => <numeric-formula> <gap> <'-'> <gap> < <numeric-term>
        #U <atomic-numeric-formula> > #.


#DF numeric-term

     => <numeric-factor>
     => <product>
     => <quotient>#.


#DF product

     => <numeric-term #U atomic-numeric-formula> <gap> <'*'>
        <gap> <numeric-factor #U atomic-numeric-formula> #.


#DF quotient

     => <numeric-term #U atomic-numeric-formula> <gap> <'/'>
        <gap> < <numeric-factor> #U <atomic-numeric-formula>
        > #.
```

```
#DF numeric-factor

     => <numeric-primary>
     => <exponential> #.


#DF exponential

     => <numeric-factor #U atomic-numeric-formula> <gap>
        <'**'> <gap> <numeric-primary #U
        atomic-numeric-formula>
     => <numeric-factor #U atomic-numeric-formula> <gap>
        <'(*'> <gap> <numeric-formula> <gap> <'*)'> #.


#DF numeric-primary

     => abs-function
     => nwdsen-function
     => loc-function
     => <'('> <gap> <numeric-formula> <gap> <')'> #.


#DF nwdsen-function

     => <'NWDSEN', '[']NWDSEN'> <gap> <'('> <gap>
        <tabular-name> <gap> <')'> #.


#DF loc-function

     => <'[']LOC'> <gap> <'('> <gap> <loc-name> <gap> <')'>
     => <'[']LOC'> <gap> <'('> <gap> <loc-name> <gap> <'.'>
        <gap> <')'> #.


#DF abs-function

     => <'ABS', '[']ABS'> <gap> <'('> <gap>
        <numeric-formula> <gap> <')'>
     => <'(/'> <gap> <numeric-formula> <gap> <'/)'> #.


#DF tabular-name

     => <name> #.
```

```
#DF loc-name

      => <name>#.

#DF atomic-numeric-formula

      => <unary-minus>
      => <unary-plus>
      => <numeric-variable>
      => <numeric-constant>
      => <function-call> #.

#DF unary-plus

      => <'+'> <gap> <atomic-numeric-formula> #.

#DF unary-minus

      => <'-'> <gap> <atomic-numeric-formula> #.

#DF function-call

      => <function-name> <gap> <'('> <gap>
         <optional-actual-input-parameter-list> <gap> <')'>
         #.

#DF function-name

      => <name>#.

#DF optional-actual-input-parameter-list

      => <#NIL>
      => <actual-input-parameter-list>#.

#DF actual-input-parameter-list

      => <actual-input-parameter>
      => <actual-input-parameter-list> <gap> <','> <gap>
```

```
        <actual-input-parameter> #.


#DF actual-input-parameter

     => <formula>
     => <actual-input-close-parameter> <gap> <'.'> #.


#DF actual-input-close-parameter

     => <name> #.


#DF formula

     => <numeric-expression> #U <boolean-expression> #U
        <atomic-formula> #.


#DF atomic-formula

     => <variable>
     => <numeric-constant> #U <string-constant> #U
        <status-constant>
     => <function-call>#.


#DF atomic-boolean-formula

     => <boolean-variable>
     => <boolean-constant>
     => <function-call>#.


#DF boolean-formula

     => <atomic-boolean-formula>
     => <boolean-expression>#.


#DF boolean-expression

     => <boolean-term>
     => <disjunction> #.


#DF disjunction
```

```
     => <disjunct> <%1 < <gap> <'OR', '[']OR'> <gap>
        <disjunct> > > #.


#DF disjunct

     => <boolean-term> #U <atomic-boolean-formula> #.


#DF boolean-term

     => <boolean-primary>
     => <conjunction> #.


#DF conjunction

     => <conjunct> <%1 < <gap> <'AND', '[']AND'> <gap>
        <conjunct> > > #.


#DF conjunct

     => <boolean-primary> #U <atomic-boolean-formula> #.


#DF boolean-primary

     => <relation-formula>
     => <negation>
     => <'('> <gap> <boolean-formula> <gap> <')'> #.


#DF negation

     => <'NOT', '[']NOT'> <gap> < <boolean-primary> #U
        <atomic-boolean-formula> > #.


#DF relation-formula

     => <chain-relation>
     => <status-relation>
     => <entry-relation> #.


#DF chain-relation
```

```
    => <relation> #.

#DF status-relation

    => <declared-variable> <gap> <relation-constant> <gap>
       <status-constant> #.

#DF entry-relation

    => <entry-variable> <gap> <equality-relation-constant>
       <gap> <entry-variable #U zero> #.

#DF equality-relation-constant

    => <'EQ', '[']EQ'>
    => <'NQ', '[']NQ'> #.

#DF zero

    => <'0'> #.

#DF relation

    => < <relational-operand> #U <relation> > <gap>
       <relation-constant> <gap> <relational-operand> #.

#DF relational-operand

    => <numeric-formula>
    => <string-constant>
    => <special-literal-variable> #.

#DF relation-constant

    => <equality-relation-constant> #U
       <inequality-relation-constant> #.

#DF inequality-relation-constant
```

```
     => <'GR', '[']GR'>
     => <'GQ', '[']GQ'>
     => <'LS', '[']LS'>
     => <'LQ', '[']LQ'> #.


#DF declared-variable

     => <simple-variable> #U <indexed-variable>#.


#DF simple-variable

     => <name>#.


#DF indexed-variable

     => <name> <gap> <'($'> <gap> <index-list> <gap> <'$)'>
        #.


#DF index-list

     => <index-formula>
     => <index-formula> <gap> <','> <gap> <index-list> #.


#DF index-formula

     => <numeric-formula> #.


#DF special-integer-variable

     => <loop-variable>
     => <'BIT', '[']BIT'> <gap> <'($'> <gap> <index-list>
        <gap> <'$)'> <gap> <'('> <gap> <declared-variable>
        <gap> <')'>
     => <'CHAR', '[']CHAR'> <gap> <'('> <gap>
        <floating-variable> <gap> <')'>
     => <'POS', '[']POS'> <gap> <'('> <gap> <file-name>
        <gap> <')'>
     => <'NENT', '[']NENT'> <gap> <'('> <gap> <table-name>
        <gap> <')'> #.


#DF table-name
```

```
    => <name>#.

#DF floating-variable

    => <declared-variable>#.

#DF file-name

    => <name>#.

#DF special-fixed-variable

    => <'MANT', '[']MANT'> <gap> <'('> <gap>
       <floating-variable> <gap> <')'> #.

#DF numeric-variable

    => <declared-variable> #U <special-integer-variable> #U
       <special-fixed-variable> #.

#DF special-literal-variable

    => <'BYTE', '[']BYTE'> <gap> <'($'> <gap> <index-list>
       <gap> <'$)'> <gap> <'('> <gap> <declared-variable>
       <gap> <')'> #.

#DF special-boolean-variable

    => <odd-keyword> <gap> <'('> <gap> <loop-variable>
       <gap> <')'>
    => <odd-keyword> <gap> <'('> <gap> <declared-variable>
       <gap> <')'> #.

#DF odd-keyword

    => <'ODD', '[']ODD'> #.

#DF boolean-variable
```

=> <declared-variable> #U <special-boolean-variable>#.

#DF entry-variable

=> <'ENTRY', '[']ENTRY', 'ENT', '[']ENT'> <gap> <'('>
<gap> <tabular-name> <gap> <'($'> <gap> <index-list>
<gap> <'$)'> <gap> <')'> #.

#DF variable

=> <declared-variable> #U <special-integer-variable> #U
<special-fixed-variable> #U
<special-boolean-variable>#U
<special-literal-variable> #U <entry-variable> #.

#DF constant

=> <floating-constant> #U <fixed-constant> #U
<integer-constant> #.

#DF optionally-signed-constant

=> <optional-sign> <numeric-constant> #.

#DF numeric-constant

=> <integer-constant> #U <floating-constant> #U
<fixed-constant> #U <octal-constant> #.

#DF optionally-signed-integer-constant

=> <optional-sign> <integer-constant> #.

#DF numeral

=> <%1 <decimal-digit>> #.

#DF integer-constant

=> <numeral>

```
     => <numeral> <'E'> <numeral> #.


#DF octal-constant

     => <'O('> <%1 <octal-digit>> <')'> #.


#DF fixed-constant

     => <simple-floating-constant> <'A'>
        <fractional-bits-designator>
     => <exponentiated-floating-constant> <'A'>
        <fractional-bits-designator> #.


#DF fractional-bits-designator

     => <#DECNUM> #.


#DF floating-constant

     => <simple-floating-constant>
     => <exponentiated-floating-constant> #.


#DF simple-floating-constant

     => <numeral> <'.'>
     => <numeral> <'.'> <numeral>
     => <'.'> <numeral> #.


#DF exponentiated-floating-constant

     => <simple-floating-constant> <'E'> <exponent> #.


#DF exponent

     => <#DECNUM> #.


#DF optional-sign

     => < '+', '-', #NIL > #.
```

#DF transmission-code-constant

=> <numeral> <'T'> <open-paren-for-constant>
<sign-string> <close-paren-for-constant> #.

#DF sign-string

=> <%1 <sign> > #.

#DF hollerith-constant

=> <numeral> <'H'> <open-paren-for-constant>
<general-sign-string> <close-paren-for-constant> #.

#DF general-sign-string

=> <%1 <general-sign> > #.

#DF general-sign

=> <sign #U system-character> #.

#DF system-character

=> <#EMPTYSET> #.

#DF status-constant

=> <'V'> <open-paren-for-constant> <#CAP #U name #U
primitive> <close-paren-for-constant> #.

#DF boolean-constant

=> <'0','1'> #.

#DF string-constant

=> <hollerith-constant> #U <transmission-code-constant>
#.

```
#DF name

     => < <#CAP> < %1 <name-letter>> > #S- <'ABS', 'ALL',
        'AND', 'ARRAY', 'ASSIGN', 'BEGIN', 'BIT', 'BYTE',
        'CHAR', 'CLOSE', 'DEFINE', 'DIRECT', 'END', 'ENT',
        'ENTRY', 'EQ', 'FILE', 'FOR', 'GOTO', 'GQ', 'GR',
        'IF', 'IFEITH', 'INPUT', 'ITEM', 'JOVIAL', '[']LOC',
        'LQ', 'LS', 'MANT', 'MODE', 'NENT', 'NOT', 'NQ',
        'NWDSEN', 'ODD', 'OPEN', 'OR', 'ORIF', 'OUTPUT',
        'OVERLAY', 'POS', 'PROC', '[']PROGRAM', 'RETURN',
        'SHUT', 'START', 'STOP', 'STRING', 'SWITCH',
        'TABLE', 'TERM', 'TEST'> #.


#DF name-letter

     => <alphameric>
     => <'[']'> <alphameric> #.


#DF octal-digit

     => <'0', '1', '2', '3', '4', '5', '6', '7'> #.


#DF decimal-digit

     => <#DIGIT> #.


#DF alphameric

     => <#CAP> #U <#DIGIT> #.


#DF prime

     => <'[']'> #.


#DF double-prime

     => <'['][']'> #.


#DF mark
```

```
     => <'+', '-', '*', '/', #SPACE, '.', '=', '(', ')',
        '$'> #U <comma> #U <prime> #.


#DF sign

     => <#CAP> #U <#DIGIT> #U <mark> #.


#DF loop-variable

     => <#CAP> #.


#DF comma

     => <','> #.


#DF list-begin-bracket

     => <begin-keyword> #.


#DF list-end-bracket

     => <end-keyword> #.


#DF gap

     => <#GAP>
     => <#GAP> <special-separators> <#GAP> #.


#DF special-separators

     => <special-separator>
     => <special-separator> <#GAP> <special-separators> #.


#DF special-separator

     => <eol>
     => <comment>
     => <special-def-mark> #.
```

```
#DF special-def-mark

     => <'[NUL]'> #.

#DF comment

     => <'[']['']'> <% <comment-char> > <'[']['']'> #.

#DF comment-char

     => <#NIL, '['']'> < <#ASCII> #S- <'['']'> > #.

#DF eol

     => <'[LF]'> #.
```

```
#DF token-list

     => <token-gap> <%<<token> <token-gap>>> #.


#DF token

     => <name-token>
     => <restricted-delimiter>
     => <numeric-form>
     => <special-constant>
     => <letter>
     => <primitive> #S- <'[']DEFINE', 'DEFINE'>
     => <define-directive> #.


#DF name-token

     => <name> #.


#DF restricted-delimiter

     => <mark> #S- <'[']', #SPACE> #.


#DF numeric-form

     => <#NAT-NOS> #.


#DF special-constant

     => <hollerith-constant>
     => <transmission-code-constant>
     => <status-constant> #.


#DF letter

     => <#CAP> #.


#DF primitive

     => <reserved-word>
```

```
     => <'[']'> <reserved-word>
     => <'[']LOC', '[']PROGRAM'> #.


#DF reserved-word

     => <'ABS', 'ALL', 'AND', 'ARRAY', 'ASSIGN', 'BEGIN',
        'BIT', 'BYTE', 'CHAR', 'CLOSE', 'DEFINE', 'DIRECT',
        'END', 'ENT', 'ENTRY', 'EQ', 'FILE', 'FOR', 'GOTO',
        'GQ', 'GR', 'IF', 'IFEITH', 'INPUT', 'ITEM',
        'JOVIAL', 'LQ', 'LS', 'MANT', 'MODE', 'NENT', 'NOT',
        'NQ', 'NWDSEN', 'ODD', 'OPEN', 'OR', 'ORIF',
        'OUTPUT', 'OVERLAY', 'POS', 'PROC', 'RETURN',
        'SHUT', 'START', 'STOP', 'STRING', 'SWITCH',
        'TABLE', 'TERM', 'TEST'> #.


#DF defined-name

     => <name> #.


#DF definiens

     => <def-mark> <restricted-token-gap>
        <%<<restricted-token> <restricted-token-gap>>>
        <def-mark> #.


#DF def-mark

     => <#NIL> #.


#DF restricted-token-gap

     => <#GAP>
     => <#GAP> <eols> <#GAP> #.


#DF eols

     => <eol> <%<<#GAP> <eol>>> #.


#DF restricted-token

     => <name-token>
```

```
        => <<restricted-delimiter> #S- <'$'>>
        => <numeric-form>
        => <restricted-special-constant>
        => <letter>
        => <primitive> #.


#DF restricted-special-constant

        => <restricted-hollerith-constant>
        => <restricted-transmission-constant>
        => <status-constant> #.


#DF restricted-transmission-constant

        => <numeral> <'T'> <open-paren-for-constant>
           <restricted-sign-string> <close-paren-for-constant>
           #.


#DF open-paren-for-constant

        => <'[STX](')> #.


#DF close-paren-for-constant

        => <')[ETX]'> #.


#DF restricted-sign-string

        => <%1 <restricted-pseudo-sign>> #.


#DF restricted-pseudo-sign

        => <restricted-sign>
        => <prime> <restricted-sign> #.


#DF restricted-sign

        => <<sign> #S- <'['']'>> #.


#DF restricted-hollerith-constant
```

```
     => <numeral> <'H'> <open-paren-for-constant>
        <restricted-general-sign-string>
        <close-paren-for-constant> #.


#DF restricted-general-sign-string

     => <%1<restricted-general-pseudo-sign>> #.


#DF restricted-general-pseudo-sign

     => <restricted-general-sign>
     => <prime> <restricted-general-sign> #.


#DF restricted-general-sign

     => <<general-sign> #S- <'[']'>> #.


#DF token-gap

     => <#GAP>
     => <#GAP> <eols-and-or-comments> <#GAP> #.


#DF eols-and-or-comments

     => <eol-or-comment>
     => <eol-or-comment> <#GAP> <eols-and-or-comments> #.


#DF eol-or-comment

     => <eol>
     => <comment> #.
```

```
#DF binary-string

     => <%1 <'0', '1'>> #.
```

```
#CONTROL-COMMANDS:

  #ASSIGN-VALUE! jovial-system = #CONTEXT-FREE-PARSE-TREE
  (textually-transformed (#GIVEN-PROGRAM), "wrt"
  <jovial-j3-system>)

  #IF ($jovial-system$)is-not-syntactically-valid
  #THEN #COMPUTE! #ERROR

  #IF there-are-executable-units-in (main-program-of
  (jovial-system) ) #THEN

  #BEGIN

    #ASSIGN-VALUE! current-executable-unit =
    first-executable-unit-in-program
    (main-program-of(jovial-system))

    #WHILE ($current-executable-unit$)is-not-terminator #DO

    #BEGIN

      #COMPUTE! computational-effect-of
      (current-executable-unit)

      #ASSIGN-VALUE! current-executable-unit =
      executable-unit-successor-of (current-executable-unit)

    #END

  #END

  #COMPUTE!  #STOP  #.
```

```
#SEMANTIC-DEFINITIONS:


#DF default-declarations-text

    => replace-all-occurrences-of ('$', "in" default-text,
    "with" '$[LF]') #.


#DF replace-all-occurrences-of (x, "in" y, "with" z)

    "{ x #IS #STRING #AND y #IS #STRING #AND z #IS #STRING
    }"

    => y #IF x #IS-NOT #SUBWORD y;

    => #PREFIX-OF-FIRST x #IN y #CW z #CW
       replace-all-occurrences-of (x, "in" #SUFFIX-OF-FIRST
       x #IN y, "with" z) #OTHERWISE #.


#DF default-text

    => 'DEFAULT DECLARATIONS:' #CW '[LF]'  #CW
       default-mode-declaration #CW '[LF]'  #CW
       default-rem-function #CW '[LF]'  #CW
       default-remquo-procedure #CW '[LF]' #.


#DF default-mode-declaration

    => 'MODE I 35 S $' #.


#DF default-rem-function

    => replace-all-occurrences-of ('delta', "in"
       standard-rem-function, "with" bits-per-word) #.


#DF standard-rem-function

    => 'PROC REM (NUM,DEN) $' #CW
          'ITEM REM I DELTA S $' #CW
          'ITEM NUM I DELTA S $' #CW
          'ITEM DEN I DELTA S $' #CW
          'BEGIN REM=NUM/DEN $' #CW
          'REM=NUM-DEN*DEN $' #CW
```

```
          'END' #.

#DF default-remquo-procedure

     => replace-all-occurrences-of ('DELTA', "in"
        standard-remquo-procedure, "with" bits-per-word) #.

#DF standard-remquo-procedure

     => 'PROC REMQUO (NUM,DEN=QUO,REM) $' #CW
           'ITEM NUM I DELTA S $' #CW
           'ITEM DEN I DELTA S $' #CW
           'ITEM QUO I DELTA S $' #CW
           'ITEM REM I DELTA S $' #CW
           'BEGIN QUO = NUM/DEN $' #CW
           'REM = NUM-QUO*DEN $' #CW
           'END' #.

#DF textually-transformed(prog)

     "{prog #EQW #GIVEN-PROGRAM }"

     => ($ ($ prog #CW default-declarations-text $)
        with-special-constants-transformed$)
        with-define-directives-applied #.

#DF with-special-constants-transformed(prog)

     "{prog #IS #STRING }"

     => prog #IF 'H(' #IS-NOT #SUBWORD prog #AND 'T('
        #IS-NOT #SUBWORD prog #AND 'V(' #IS-NOT #SUBWORD
        prog ;

     => ($prog ,"with-forms-beginning-at"
        position-of-first-form-in(prog)
        ,"to"position-last-char-of-first-form-in (prog)$)
        rewritten #OTHERWISE #.

#DF position-of-first-form-in(text)

     "{ text #IS #STRING }"
```

```
      => #FIRST char-pos : 1 <= char-pos <= #LENGTH (text)+1
         #SUCH-THAT ( ($char-pos ,"in" text$) starts-a-form
         #OR char-pos = #LENGTH (text)+1) #.


#DF starts-a-form(char-pos
         ,"in" text)

      "{ text #IS #STRING #AND 1 < = char-pos #AND char-pos
      <= #LENGTH (text) + 1}"

      => #FALSE #IF char-pos >= #LENGTH (text);

      => #TRUE #IF ( #SUBSTRING-OF-CHARACTERS char-pos #TO
         char-pos + 1 #OF text) #EQW '[']['']' ;

      => ($ #SUBSTRING-OF-CHARACTERS char-pos #TO #LENGTH
         (text) #OF text$) begins-as-a-special-constant-would
         #IF ($char-pos ,"th-character-in" text $)
         has-allowable-left-context;

      => #FALSE #OTHERWISE #.


#DF has-allowable-left-context(char-pos
         ,"th-character-in" text)

      "{ text #IS #STRING #AND 1<=char-pos #AND char-pos<
      #LENGTH (text)}"

      => #TRUE #IF char-pos = 1;

      => #TRUE #IFF (char-pos - 1) #TH-CHARACTER-IN text #IS
         #SUBWORD ('()+-*1.,=$[']' #CW #SPACE #CW '[LF]')
         #OTHERWISE #.


#DF begins-as-a-special-constant-would( text)

      "{ text #IS #STRING }"

      => #FALSE #IF #LENGTH (text) <4;

      => #TRUE #IF #LEFT 2 #CHARACTERS-OF text #EQW 'V(';

      => #FALSE #IF 'H(' #IS-NOT #SUBWORD text #AND 'T('
         #IS-NOT #SUBWORD text ;
```

```
    => #TRUE #IFF ($ #PREFIX-OF-FIRST '(' #IN text $)
       is-initial-part-of-literal-constant #OTHERWISE #.


#DF is-initial-part-of-literal-constant(text)

    "{ text #IS #STRING }"

    => #FALSE #IF #LENGTH (text) < 2 ;

    => #FALSE #IF #LAST-CHARACTER-IN text #IS-NOT-IN \'H'
       ,'T'\;

    => #TRUE #IFF #FOR-ALL i : 1 <= i <=( #LENGTH (text) -
       1) #IT-IS-TRUE-THAT ( i #TH-CHARACTER-IN text #IS
       #DIGIT ) #OTHERWISE #.


#DF position-last-char-of-first-form-in(prog)

    "{ prog #IS #STRING }"

    => last-char-posn-in (prog ,"given"
       position-of-first-form-in(prog)) #.


#DF last-char-posn-in(text
       ,"given" first-char-pos)

    "{text #IS #STRING #AND 1 <= first-char-pos #AND
    first-char-pos <= #LENGTH (text) +1 }"

    => first-char-pos #IF first-char-pos = #LENGTH (text)
       +1;

    => last-comment-char-posn-in (text ,"given"
       first-char-pos) #IF first-char-pos #TH-CHARACTER-IN
       text #EQW '[']';

    => last-status-constant-char-posn-in (text ,"given"
       first-char-pos) #IF first-char-pos #TH-CHARACTER-IN
       text #EQW 'V';

    => last-literal-constant-char-posn-in (text ,"given"
       first-char-pos) #IF first-char-pos #TH-CHARACTER-IN
       text #IS #DIGIT #.
```

```
#DF last-comment-char-posn-in(text
        ,"given" first-char-pos)

     "{ text #IS #STRING #AND 1<=first-char-pos #AND
     first-char-pos <= #LENGTH (text)}"

     => #LENGTH (text)+1 #IF #LENGTH (text)<first-char-pos
        +3 ;

     => #FIRST char-pos :  (first-char-pos+3)<=char-pos <=
        #LENGTH (text)+1 #SUCH-THAT ( ($char-pos ,"in"text$)
        terminates-comment-form #OR char-pos = #LENGTH
        (text)+1) #OTHERWISE #.


#DF terminates-comment-form(char-pos
        ,"in" text)

     "{text #IS #STRING #AND 4<=char-pos #AND char-pos<=
     #LENGTH (text)+1}"

     => #FALSE #IF char-pos = #LENGTH (text)+1;

     => #TRUE #IFF #SUBSTRING-OF-CHARACTERS char-pos - 1 #TO
        char-pos #OF text #EQW '['][']' #OTHERWISE #.


#DF last-status-constant-char-posn-in(text
        ,"given" first-char-pos)

     "{text #IS #STRING #AND 1<=first-char-pos #AND
     first-char-pos<= #LENGTH (text)}"

     => #LENGTH (text) +1 #IF #LENGTH (text) <
        first-char-pos + 3 ;

     => #LENGTH (text) +1 #IF (first-char-pos+1)
        #TH-CHARACTER-IN text #NEQW '(' #OR (first-char-pos
        +2) #TH-CHARACTER-IN text #IS-NOT #CAP ;

     => #FIRST char-pos :  (first-char-pos+3) <= char-pos <=
        #LENGTH (text)+1 #SUCH-THAT ( ($char-pos ,"in"
        text$) terminates-status-constant-form #OR char-pos
        = #LENGTH (text)+1) #OTHERWISE #.


#DF terminates-status-constant-form(char-pos
        ,"in" text)
```

```
    "{text #IS #STRING #AND 4<=char-pos #AND char-pos <=
    #LENGTH (text)+1}"

    => #FALSE #IF char-pos = #LENGTH (text)+1;

    => #TRUE #IFF char-pos #TH-CHARACTER-IN text #EQW ')'
       #OTHERWISE #.


#DF last-literal-constant-char-posn-in(text
       ,"given" first-char-pos)

    "{text #IS #STRING #AND 1<=first-char-pos #AND
    first-char-pos < #LENGTH (text)}"

    => ($last-char-posn-determined-by(text ,"starting-at"
       first-char-pos) ,"against" text$)
       validated-for-consistency #.


#DF last-char-posn-determined-by(text
       ,"starting-at" first-char-pos)

    "{text #IS #STRING #AND 1<=first-char-pos #AND
    first-char-pos <= #LENGTH (text)}"

    => first-char-pos +
       literal-length-determined-by(initial-number-part-of
       (text ,"starting-at" first-char-pos)) - 1 #.


#DF initial-number-part-of(text
       ,"starting-at" first-char-pos)

    "{text #IS #STRING #AND 1<=first-char-pos #AND
    first-char-pos <= #LENGTH (text)}"

    => all-but-last-character-of ( #PREFIX-OF-FIRST '(' #IN
       characters-after (first-char-pos - 1 ,"in" text)) #.


#DF all-but-last-character-of(str)

    "{str #IS #STRING }"

    => #LEFT ( #LENGTH (str) - 1) #CHARACTERS-OF str #.
```

```
#DF literal-length-determined-by( nr-of-chars)

     "{nr-of-chars #IS #STRING #AND nr-of-chars #IS #INTEGER
     }"

     => #LENGTH (nr-of-chars) + #LENGTH ('H(')+ nr-of-chars
        + #LENGTH (')') #.


#DF validated-for-consistency( last-char-pos
        ,"against" text)

     "{text #IS #STRING #AND last-char-pos >4}"

     => #LENGTH (text)+1 #IF #LENGTH (text) < last-char-pos;

     => #LENGTH (text)+1 #IF last-char-pos #TH-CHARACTER-IN
        text #NEQW ')' ;

     => last-char-pos #OTHERWISE #.


#DF rewritten(prog
        ,"starting-with-form-at" first-char-pos ,"to"
        last-char-pos)

     "{ prog #IS #STRING #AND first-char-pos <=
     last-char-pos}"

     => prog #IF ($first-char-pos ,"in" prog$)
        specifies-no-form-at-all;

     => incomplete-form-error-at ( first-char-pos ,"in" prog
        ) #IF ($ last-char-pos ,"in" prog$)
        specifies-an-incomplete-form;

     => characters-up-to ( first-char-pos ,"in" prog) #CW
        modified(special-form-at (first-char-pos ,"to"
        last-char-pos ,"in" prog)) #CW
        ($characters-after(last-char-pos ,"in" prog)$)
        with-special-constants-transformed #IF
        ($last-char-pos ,"and" first-char-pos ,"in" prog$)
        specify-a-complete-form #.


#DF specifies-no-form-at-all(char-pos
        ,"in" prog)
```

```
     "{ char-pos #IS #INTEGER #AND prog #IS #STRING }"

     => #TRUE #IFF char-pos < 1 #OR #LENGTH (prog) <
        char-pos #.


#DF specifies-an-incomplete-form(char-pos
        ,"in" prog)

     "{char-pos #IS #INTEGER #AND prog #IS #STRING }"

     => #TRUE #IFF char-pos < 1 #OR #LENGTH (prog) <char-pos
        #.


#DF specify-a-complete-form(last-char-pos
        ,first-char-pos ,"in" prog)

     "{first-char-pos #IS #INTEGER #AND last-char-pos #IS
     #INTEGER #AND prog #IS #STRING }"

     => #TRUE #IFF 1 <= first-char-pos #AND first-char-pos
        <= last-char-pos #AND last-char-pos <= #LENGTH
        (prog) #.


#DF characters-up-to(char-pos
        ,"in" prog)

     "{prog #IS #STRING #AND 1 <= char-pos #AND char-pos <=
     #LENGTH (prog)}"

     => #LEFT (char-pos - 1) #CHARACTERS-OF prog #.


#DF characters-after(char-pos
        ,"in" prog)

     "{prog #IS #STRING #AND 1 <= char-pos #AND char-pos <=
     #LENGTH (prog)}"

     => #RIGHT ( #LENGTH (prog) - char-pos) #CHARACTERS-OF
        prog #.


#DF special-form-at(pos1
        ,"to" pos2 ,"in" prog)
```

```
    "{prog #IS #STRING #AND pos1 < pos2 #AND pos2 <=
    #LENGTH (prog)}"

    => #SUBSTRING-OF-CHARACTERS pos1 #TO pos2 #OF prog #.

#DF modified(form)

    "{ ($form$) is-special-form}"

    => '['][']' #CW  transformed-comment-or-definiens
       (text-of (form)) #CW '['][']' #IF
       #FIRST-CHARACTER-IN form #EQW '[']';

    => transformed-special-constant (form) #OTHERWISE #.

#DF transformed-special-constant(form)

    "{ ($form$) is-special-constant-form}"

    => ( #PREFIX-OF-FIRST '(' #IN form ) #CW '[STX](' #CW (
       #SUFFIX-OF-FIRST '(' #IN form) #CW '[ETX]' #.

#DF text-of( form)

    "{ ($form$) is-comment-like-special-form}"

    => #NIL #IF #LENGTH ( form) =4;

    => #SUBSTRING-OF-CHARACTERS 3 #TO ( #LENGTH ( form) -
       2) #OF form #OTHERWISE #.

#DF transformed-comment-or-definiens(text)

    "{ text #IS #STRING }"

    => text #IF 'H(' #IS-NOT #SUBWORD text #AND 'T('
       #IS-NOT #SUBWORD text #AND 'V(' #IS-NOT #SUBWORD
       text ;

    => ($text ,"with-forms-beginning-at"
       position-of-first-form-in (text)
       ,"to"position-last-char-of-first-form-in (text)$)
       rescribed #OTHERWISE #.
```

```
#DF rescribed(text
        ,"starting-with-format" first-char-pos ,"to"
        last-char-pos)

     "{text #IS #STRING #AND first-char-pos <=
     last-char-pos}"

     => text #IF ($first-char-pos ,"in" text$)
        specifies-no-form-at-all;

     => text #IF ($last-char-pos ,"in" text$)
        specifies-an-incomplete-form;

     => characters-up-to (first-char-pos ,"in" text) #CW
        modified (special-form-at (first-char-pos ,"to"
        last-char-pos ,"in" text)) #CW
        transformed-comment-or-definiens(characters-after
        (last-char-pos ,"in" text)) #IF ($last-char-pos
        ,"and" first-char-pos ,"in" text$)
        specify-a-complete-form #.

#DF with-define-directives-applied(prog)

     "{prog #IS #STRING }"

     => prog #IF 'DEFINE' #IS-NOT #SUBWORD prog;

     => ($ ($prog$) parsed-as-a-token-string$)
        processed-wrt-defines #OTHERWISE #.

#DF parsed-as-a-token-string(prog)

     "{prog #IS #STRING }"

     => #CONTEXT-FREE-PARSE-TREE (prog ,"wrt" <token-list>)
        #.

#DF processed-wrt-defines(tlist)

     "{tlist #IS <token-list> #OR tlist #IS #UNDEFINED }"

     => #NIL #IF tlist #IS #UNDEFINED ;
```

```
     => ($ ($sequence-of-program-level-tokens-and-gaps-in
        (tlist)$) with-defines-expanded$)
        converted-to-string-form #OTHERWISE #.


#DF sequence-of-program-level-tokens-and-gaps-in(tlist)

     "{tlist #IS <token-list>}"

     => #SUBSEQUENCE-OF-ELEMENTS x #IN (#SEQUENCE-OF
        <name-token> #U <restricted-delimiter> #U
        <numeric-form> #U <special-constant> #U <letter> #U
        <primitive> #U <define-directive> #U <token-gap> #IN
        tlist ) #SUCH-THAT (#FOR-ALL y #IN
        #SEQUENCE-OF-ANCESTORS-OF x #IT-IS-TRUE-THAT ( y
        #IS-NOT <define-directive> )) #.


#DF converted-to-string-form(seq)

     "{seq #EQ transformed-token-seq}"

     => #NIL #IF seq #EQ #NILSEQ ;

     => string-corresponding-to ( #FIRST-ELEMENT-IN seq )
        #CW ($all-but-first-element-in (seq)$)
        converted-to-string-form #OTHERWISE #.


#DF string-corresponding-to(tok)

     "{tok #IS-IN transformed-token-seq}"

     => special-def-mark-character #IF tok #IS <def-mark>;

     => #STRING-OF-TERMINALS-OF (tok) #OTHERWISE #.


#DF special-def-mark-character

     => '[NUL]' #.


#PROC-DF with-defines-expanded(tseq)

   #ASSIGN-VALUE!  unscanned-token-seq = tseq

   #ASSIGN-VALUE!  transformed-token-seq = #NILSEQ
```

```
  #WHILE unscanned-token-seq #NEQ #NILSEQ #DO

  #COMPUTE!  transformation-induced-by (next-token-in
  (unscanned-token-seq))

  #RETURN-WITH-VALUE!  transformed-token-seq #.


#DF next-token-in(seq)

     "{ seq #EQ unscanned-token-seq}"

     => #FIRST-ELEMENT-IN seq #.


#DF transformation-induced-by(next-tok)

     "{next-tok #EQ next-token-in (unscanned-token-seq)}"

     => expand (next-tok) #IF ($next-tok$)
        is-a-defined-name;

     => pass-across(next-tok) #OTHERWISE #.


#PROC-DF pass-across(next-tok)

     "{ ($next-tok$) is-a-defined-name #EQ #FALSE }"

     #ASSIGN-VALUE!  unscanned-token-seq =
     all-but-next-tok-in(unscanned-token-seq)

     #ASSIGN-VALUE!  transformed-token-seq =
     transformed-token-seq #CS \next-tok\

     #RETURN-WITH-VALUE!  #NIL #.


#DF all-but-next-tok-in(seq)

     "{ seq #EQ unscanned-token-seq}"

     => all-but-first-element-in (seq) #.


#DF is-a-defined-name(next-tok)
```

```
    "{ next-tok #EQ next-token-in (unscanned-token-seq)}"

    => #FALSE #IF next-tok #IS-NOT <name-token> ;

    => #TRUE #IFF #THERE-EXISTS def-dir #IN
       subsequence-of-define-directives-contained-in
       (transformed-token-seq) #SUCH-THAT
       (define-name-declared-in(def-dir) #EQW next-tok)
       #OTHERWISE #.


#DF subsequence-of-define-directives-contained-in(seq)

    "{ seq #EQ transformed-token-seq }"

    => #SUBSEQUENCE-OF-ELEMENTS tok #IN seq #SUCH-THAT (
       tok #IS <define-directive>) #.


#DF define-name-declared-in(def-dir)

    "{def-dir #IS <define-directive>}"

    => #SEG 3 #OF def-dir #.


#PROC-DF expand(next-tok)

    "{ ($next-tok$) is-a-defined-name}"

    #ASSIGN-VALUE!  unscanned-token-seq =
    token-seq-defined-for(next-tok) #CS all-but-next-tok-in
    (unscanned-token-seq)

    #RETURN-WITH-VALUE!  #NIL #.


#DF token-seq-defined-for(next-tok)

    "{ next-tok #EQ next-token-in (unscanned-token-seq)}"

    => sequence-of-definiens-level-tokens-and-gaps-in
       (definiens-of (define-directive-defining
       (next-tok))) #.


#DF define-directive-defining(next-tok)
```

```
     "{ next-tok #EQ next-token-in (unscanned-token-seq)}"

     => #LAST def-dir #IN
        subsequence-of-define-directives-contained-in
        (transformed-token-seq) #SUCH-THAT
        (define-name-declared-in (def-dir) #EQW next-tok )
        #.


#DF definiens-of(def-dir)

     "{def-dir #IS <define-directive>}"

     => #SEG 6 #OF def-dir #.


#DF sequence-of-definiens-level-tokens-and-gaps-in(def)

     "{def #IS <definiens>}"

     => #SEQUENCE-OF <name-token> #U <restricted-delimiter>
        #U <numeric-form> #U <letter> #U <primitive> #U
        <restricted-special-constant> #U <def-mark> #U
        <restricted-token-gap> #IN def #.


#DF incomplete-form-error-at (char-pos ,"in" text)

     "{text #IS #STRING #AND 1 <= char-pos #AND char-pos <=
     #LENGTH (text)}"

     => fatal-lexical-error
        ('malformed-special-form-at-underlined-char:' #CW
        eol-char #CW characters-up-to (char-pos ,"in" text)
        #CW (char-pos #TH-CHARACTER-IN text) #CW '[BS]_' #CW
        characters-after (char-pos ,"in" text)) #.


#PROC-DF fatal-lexical-error (msg)

     "{msg #IS #STRING}"

     #COMPUTE! #OUTPUT (msg #CW eol-char)

     #COMPUTE! #ERROR #.


#DF eol-char
```

```
=> '[LF]' #.
```

```
#DF is-not-syntactically-valid (system)

     "{ system #IS <jovial-j3-system> #OR system #IS
     #UNDEFINED}"

     => #TRUE #IF system #IS #UNDEFINED;

     => #FALSE #IFF
        ($system$)is-contextually-syntactically-valid
        #OTHERWISE #.


#DF is-contextually-syntactically-valid(prog)

     "{ prog #IS <jovial-j3-program> }"

     => #FALSE #IF non-context-free-error-is-present-in
        (prog);

     => #TRUE #OTHERWISE #.


#PROC-DF non-context-free-error-is-present-in (prog)

  #ASSIGN-VALUE! ncf-error-is-discovered = #FALSE

  #FOR-ALL dest-name #IN dest-name-seq (prog) #DO #COMPUTE!
  context-sensitive-tests-of-sequence-designator (dest-name)

  #COMPUTE!
  context-sensitive-tests-of-returns-and-for-clauses-of
  (prog)

  #COMPUTE! context-sensitive-test-of-odd-modifiers-in
  (prog)

  #RETURN-WITH-VALUE! ncf-error-is-discovered #.


#DF dest-name-seq (prog)

     => #SEQUENCE-OF <destination-name> #IN prog #.

     "CED 2444.2: Go-to sequence-designator must be the name
     of a statement, program, close, or switch."
```

```
     "This does not check whether abnormal procedure exits
     are semantically correct; e.g., whether the go-to of
     such an exit references a statement in another
     procedure not yet activated. This, however, is a
     dynamic error in some cases."


#DF context-sensitive-tests-of-sequence-designator (dn)

     "{dn #IS <destination-name>}"

     => error-message ('CED-2444.2-VIOLATED:' #CW dn #CW
        'IS-NOT-NAME-OF-STATEMENT,-PROGRAM,-CLOSE-OR-SWITCH')
        #IF there-is-no-program-point-designated-by (dn);

     => \ test-for-illegal-loop-entry (dn),
        test-if-is-program-name-in-switch (dn) \ #OTHERWISE
        #.


#DF test-for-illegal-loop-entry (dn)

     "{dn #IS <destination-name>}"

     =>#NIL #IF point-designated-by (dn) #IS
        <formal-input-parameter> #U
        <formal-output-parameter>;

     => test-for-loop-containing (dn) #OTHERWISE #.


#DF test-for-loop-containing (dn)

     => #NIL #IF ($point-designated-by (dn)$)
        is-not-in-loop;

     => test-for-outside-loop-entry (dn) #OTHERWISE #.


#DF test-for-outside-loop-entry (dn)

     => #NIL #IF loop ( innermost-loop-containing
        (point-designated-by (dn)), "contains" dn) ;

     => error-message ('CED-2456.3-VIOLATED:' #CW dn #CW
        'IS-OUTSIDE-LOOP-CONTAINING' #CW point-designated-by
        (dn)) #OTHERWISE #.
```

```
#DF test-if-is-program-name-in-switch (dn)

     => test-if-in-switch (dn) #IF ($dn$) is-program-name;

     => #NIL #OTHERWISE #.


#DF test-if-in-switch (dn)

     => error-message ('CED-2481.2-VIOLATED:' #CW dn #CW
        'IS-PROGRAM-NAME-IN-SWITCH') #IF #THERE-EXISTS sw
        #IN #SEQUENCE-OF-ANCESTORS-OF dn  #SUCH-THAT (sw #IS
        <switch-declaration>);

     => #NIL #OTHERWISE #.


#DF point-designated-by (dn)

     => category-2-declaration-for (dn) #.


#DF there-is-no-program-point-designated-by (dn)

     =>  #NOT a-category-2-declaration-exists-for (dn) #.


#DF is-not-in-loop (dn)

     => #NOT #THERE-EXISTS x #IN #SEQUENCE-OF-ANCESTORS-OF
        dn #SUCH-THAT ( x #IS <loop-statement>) #.


#DF loop (lp ,"contains" dn)

     => lp #IS-IN #SEQUENCE-OF-ANCESTORS-OF dn #.


#DF innermost-loop-containing (pt)

     => #LAST lp #IN (#SEQUENCE-OF-ANCESTORS-OF pt)
        #SUCH-THAT (lp #IS <loop-statement>) #.


#DF is-program-name (dn)

     => point-designated-by (dn) #IS <program-name> #.
```

```
#DF error-message (string)

     => \ #OUTPUT (string #CW end-of-line-char),
        record-ncf-error\ #.


#DF end-of-line-char

     => '[LF]' #.


#PROC-DF record-ncf-error

  #ASSIGN-VALUE! ncf-error-is-discovered = #TRUE

  #RETURN-WITH-VALUE! #NIL #.


#DF context-sensitive-tests-of-returns-and-for-clauses-of
        (prog)

     => \ test-if-returns-are-all-in-proc-decls-of (prog) ,
        test-if-names-in-for-clauses-are-in-tables-of
        (prog),
        test-if-distinct-loop-vrbls-in-nested-loops-of
        (prog) \ #.
```

"CED 2445.3. A return-statement may appear only in a processing declaration."

```
#DF test-if-returns-are-all-in-proc-decls-of (prog)

     => #NIL #IF #FOR-ALL return-stmt #IN
        sequence-of-returns-in (prog) #IT-IS-TRUE-THAT
        (($return-stmt, "of" prog$) is-in-some-proc-decl);

     => error-message ('RETURN-OCCURS-IN-MAIN-PROGRAM')
        #OTHERWISE #.


#DF is-in-some-proc-decl (stmt, prog)

     => #THERE-EXISTS proc-decl #IN sequence-of-proc-decl-in
        (prog) #SUCH-THAT (($stmt,"in" proc-decl$)occurs)#.
```

```
#DF occurs (nx ,"in" ny)

     => ny #IS-IN #SEQUENCE-OF-ANCESTORS-OF nx #.

#DF sequence-of-returns-in (prog)

     => #SEQUENCE-OF <return-statement> #IN prog #.

#DF sequence-of-proc-decl-in (prog)

     => #SEQUENCE-OF <process-declaration> #IN prog#.

     "CED 2455.2 The name in a complete for clause must be a
     table name or the name of an item belonging to a
     table."

#DF test-if-names-in-for-clauses-are-in-tables-of (prog)

     => #NIL #IF #FOR-ALL for-clause #IN
        sequence-of-for-clauses-in (prog) #IT-IS-TRUE-THAT
        (($ name-in (for-clause), "in" prog $)
        is-tabular-name-or-nil) ;

     => error-message (
        'FOR-CLAUSE-HAS-A-NAME-NOT-BELONGING-TO-A-TABLE' )
        #OTHERWISE #.

#DF is-tabular-name-or-nil (nm, "in" prog)

     => #TRUE #IF nm #EQW #NIL ;

     => #TRUE #IF ($ nm, "in" prog $) is-tabular-name ;

     => #FALSE #OTHERWISE #.

#DF is-tabular-name (nm, "in" prog)

     => #THERE-EXISTS table-decl #IN (#SEQUENCE-OF
        <table-declaration> #IN prog) #SUCH-THAT
        (($declaration-for (nm) , "in" table-decl $) occurs)
        #.
```

```
#DF name-in (for-clause)

     => #NIL #IF for-clause #IS #CASE 1 #OF
        <complete-for-clause> ;

     => #SEG 1 #OF (#SEG 11 #OF for-clause) #IF for-clause
        #IS #CASE 2 #OF <complete-for-clause> #.


#DF sequence-of-for-clauses-in (prog)

     => #SEQUENCE-OF <complete-for-clause> #IN prog #.

     "CED 2456.2 A for-clause may be used to activate only a
     loop variable that is not already active."


#DF test-if-distinct-loop-vrbls-in-nested-loops-of (prog)

     => #NIL #IF #FOR-ALL for-clause #IN (#SEQUENCE-OF
        <one-factor-for-clause> #U <two-factor-for-clause>
        #U <complete-for-clause> #IN prog) #IT-IS-TRUE-THAT
        (there-is-no-previous-activation-of (loop-var-of
        (for-clause), "in"
         prog));

     => error-message
        ('DUPLICATION-OF-LOOP-VARIABLES-IN-OVERLAPPING-LOOP-STATEMENTS')
        #OTHERWISE #.


#DF there-is-no-previous-activation-of (lvar, "in" prog)

     => #FOR-ALL for-clause #IN #SEQUENCE-OF
        <one-factor-for-clause> #U <two-factor-for-clause>
        #U <complete-for-clause> #IN prog #IT-IS-TRUE-THAT (
        activation-of (loop-var-of (for-clause),"precedes"
        lvar,"in" prog) #IMPLIES loop-var-of (for-clause)
        #NEQW lvar) #.


#DF activation-of (lp1, "precedes"lp2, "in" prog)

     => ($lp1, "and"lp2$)have-common-loop #AND lp1 #PRECEDES
        lp2 #IN prog #.
```

```
#DF have-common-loop (lp-var1,"and" lp-var2)

     => #THERE-EXISTS stmt #IN #SEQUENCE-OF-ANCESTORS-OF
        lp-var1 #SUCH-THAT ( stmt #IS <loop-statement> #AND
        stmt #IS-IN #SEQUENCE-OF-ANCESTORS-OF lp-var2) #.


#DF loop-var-of (for-clause)

     => #SEG 3 #OF for-clause #.

     "CED 2429.2: ODD must not be applied to a floating
     variable."


#DF context-sensitive-test-of-odd-modifiers-in (prog)

     => #NIL #IF #FOR-ALL odd-modifier #IN
        sequence-of-odd-modifiers-in (prog) #IT-IS-TRUE-THAT
        (($ odd-modifier $) is-not-floating) ;

     => error-message (
        'CED-2429.2-VIOLATED:-ODD-IS-APPLIED-TO-A-FLOATING-VARIABLE')
        #OTHERWISE #.


#DF is-not-floating (om)

     => #TRUE #IFF type (object-variable-of (om)) #NEQW
        'floating' #.


#DF sequence-of-odd-modifiers-in (prog)

     => #SEQUENCE-OF <special-boolean-variable> #IN prog #.
```

```
#DF main-program-of (system)

     "{ system #IS <jovial-j3-system>}"

     => #SEG 2 #OF system #.


#DF there-are-executable-units-in(prog)

     "{ prog #IS <jovial-j3-program> }"

     => #TRUE #IFF sequence-of-executable-units-in(prog)
        #NEQ #NILSEQ #.


#DF sequence-of-executable-units-in (nx)

     "{nx #IS <jovial-j3-program> #U <close-subprogram> #U
     <independent-statement> #U <alternative-statement> #U
     <conditional-statement> #U <special-compound> #U
     <loop-statement} #U <unnamed-statement> #U
     <inner-close-body> #U <procedure-declaration> }"

     => sequence-of-close-subprogram-units-in (nx) #IF nx
        #IS <close-subprogram> ;

     => sequence-of-units-implied-in
        (sequence-of-outer-executable-statements-in (nx))
        #OTHERWISE #.
```

"The modifier 'implied' requires a sequence of nodes as arguments. it suggests that the lists of nodes is expanded due to the nesting of the language, each node into a series representing the nesting of statements within the structure of other statements."

```
#DF sequence-of-close-subprogram-units-in (nx)

     "{ nx #IS <close-subprogram> }"

     => sequence-of-close-body-units-in (close-body-of (nx)
        ) #.
```

```
#DF sequence-of-outer-executable-statements-in (nx)

     "{nx #IS <jovial-j3-program> #U <close-subprogram> #U
     <independent-statement> #U <alternative-statement> #U
     <conditional-statement> #U <special-compound> #U
     <loop-statement> #U <unnamed-statement> #U
     <inner-close-body> #U <procedure-declaration> }"

     => #SUBSEQUENCE-OF-ELEMENTS stmt #IN
        sequence-of-executable-statements-in (nx) #SUCH-THAT
        (($ stmt,"in" nx $) is-first-level-statement) #.
```

"The modifier 'outer' indicates that the sequence of nodes is that of a single level not including any nodes lower on the parse tree."

```
#DF sequence-of-units-implied-in (stmt-seq)

     => #NILSEQ #IF stmt-seq #EQ #NILSEQ;

     => sequence-of-units-in-statement (#FIRST-ELEMENT-IN
        stmt-seq) #CS sequence-of-units-implied-in
        (all-but-first-element-in (stmt-seq)) #OTHERWISE #.

#DF sequence-of-executable-statements-in (px)

     "{px #IS <program> #U <close-subprogram> #U
     <independent-statement> #U <alternative-statement> #U
     <conditional-statement> #U <special-compound> #U
     <loop-statement> #U <unnamed-statement> #U
     <inner-close-body> #U <procedure-declaration> }"

     => #SEQUENCE-OF <assignment-statement> #U
        <exchange-statement> #U <go-to-statement> #U
        <return-statement> #U <stop-statement> #U
        <procedure-call-statement> #U <input-statement> #U
        <output-statement> #U <open-input-statement> #U
        <open-output-statement> #U <shut-input-statement> #U
        <shut-output-statement> #U <direct-statement> #U
        <conditional-statement> #U <alternative-statement>
        #U <loop-statement> #U <test-statement> #U
        <special-test-statement> #U <term-statement> #U
        <index-switch-declaration> #U
        <item-switch-declaration> #U <close-declaration> #IN
        px #.
```

```
#DF is-first-level-statement (stmt, "in" nx)

     => #TRUE #IFF #FOR-ALL outer-stmt #IN #SEQUENCE-OF
        <compound-statement> #U <complex-statement> #U
        <close-declaration> #U <procedure-declaration> #IN
        nx #IT-IS-TRUE-THAT (outer-stmt #IS-NOT-IN
        #SEQUENCE-OF-ANCESTORS-OF (stmt) ) #.


#DF sequence-of-units-in-statement (stmt)

     => sequence-of-assignment-statement-units-in (stmt) #IF
        stmt #IS <assignment-statement> ;

     => sequence-of-exchange-statement-units-in (stmt) #IF
        stmt #IS <exchange-statement> ;

     => sequence-of-go-to-statement-units-in (stmt) #IF stmt
        #IS <go-to-statement> ;

     => sequence-of-return-statement-units-in (stmt) #IF
        stmt #IS <return-statement> ;

     => \stmt\ #IF stmt #IS <stop-statement> ;

     => \stmt\ #IF stmt #IS <procedure-end> ;

     => sequence-of-procedure-call-statement-units-in (stmt)
        #IF stmt #IS <procedure-call-statement> ;

     => sequence-of-input-statement-units-in (stmt) #IF stmt
        #IS <input-statement> ;

     => sequence-of-output-statement-units-in (stmt) #IF
        stmt #IS <output-statement> ;

     => sequence-of-open-input-statement-units-in (stmt) #IF
        stmt #IS <open-input-statement> ;

     => sequence-of-open-output-statement-units-in (stmt)
        #IF stmt #IS <open-output-statement> ;

     => sequence-of-shut-input-statement-units-in (stmt) #IF
        stmt #IS <shut-input-statement> ;

     => sequence-of-shut-output-statement-units-in (stmt)
```

```
        #IF stmt #IS <shut-output-statement> ;

     => \stmt\ #IF stmt #IS <direct-statement> ;

     => sequence-of-alternative-statement-units-in (stmt)
        #IF stmt #IS <alternative-statement>;

     => sequence-of-conditional-units-in (stmt) #IF stmt #IS
        <conditional-statement>;

     => sequence-of-loop-units-in (stmt) #IF stmt #IS
        <loop-statement>;

     => sequence-of-test-units-in (stmt) #IF stmt #IS
        <test-statement> #U <special-test-statement> ;

     => \stmt\ #IF stmt #IS <term-statement> ;

     => sequence-of-index-switch-declaration-units-in (stmt)
        #IF stmt #IS <index-switch-declaration> ;

     => sequence-of-item-switch-declaration-units-in (stmt)
        #IF stmt #IS <item-switch-declaration> ;

     => sequence-of-procedure-declaration-units-in (stmt)
        #IF stmt #IS <procedure-declaration>;

     => sequence-of-close-declaration-units-in (stmt) #IF
        stmt #IS <close-declaration> #.


#DF all-but-first-element-in (seq)

     "{seq #IS #SEQUENCE & #LENGTH (seq) > 0}"

     => #TERMINAL-SUBSEQ-OF-LENGTH (#LENGTH(seq) - 1) #OF
        seq #.


#DF first-executable-unit-in-program (prog)

     "{ prog #EQ <jovial-j3-program>}"

     => target-of (initial-statement-name-of(prog)) #IF
        ($prog$) has-an-initial-statement-name;

     => #FIRST-ELEMENT-IN sequence-of-executable-units-in
        (jovial-system) #OTHERWISE #.
```

```
#DF has-an-initial-statement-name(prog)

     "{prog #IS <jovial-j3-program>}"

     => #TRUE #IFF
        optional-initial-statement-name-of(term-statement-of
        (program-body-of(prog))) #NEQW #NIL #.

#DF initial-statement-name-of (nx)

     "{ nx #IS <jovial-j3-program> #U
     <optional-initial-statement-name>}"

     => #SEG 1 #OF nx #IF nx #IS
        <optional-initial-statement-name>;

     =>
        initial-statement-name-of(optional-initial-statement-name-of
        (term-statement-of(program-body-of(nx))))#IF nx #IS
        <jovial-j3-program> #.

#DF optional-initial-statement-name-of(term-stmt)

     "{term-stmt #IS <term-statement>}"

     => #SEG 3 #OF term-stmt #.

#DF term-statement-of(prog)

     "{prog #IS <program>}"

     => #SEG 5 #OF prog #.

#DF program-body-of (prog)

     "{prog #IS <jovial-j3-program>}"

     => #SEG 2 #OF prog #.

#DF is-not-terminator(nx)
```

```
    "{ nx #EQ current-executable-unit }"

    => #TRUE #IFF nx #IS-NOT <stop-statement> #U
       <term-statement> #.

#DF computational-effect-of (unit)

    "{unit #EQ current-executable-unit}"

    => assignment-statement-effect-of(unit) #IF unit #IS
       <assignment-statement>;

    => exchange-statement-effect-of (unit) #IF unit #IS
       <exchange-statement>;

    => procedure-call-effect-of (unit) #IF unit #IS
       <procedure-call-statement> ;

    => procedure-return-effect-of (unit) #IF ($unit$)
       is-a-return-from-a-procedure;

    => return-statement-effect-of (unit) #IF unit #IS
       <return-statement>;

    => #NIL #IF ($unit$)
       is-unit-unique-to-alternative-statement #OR ($unit$)
       is-unit-unique-to-conditional-statement #OR ($unit$)
       is-branch-control-unit-of-loop-statement #OR ($ unit
       $) is-unique-to-a-close #OR ($ unit $)
       is-unit-unique-to-go-to-statement ;

    => loop-statement-effect-of (unit) #IF ($unit$)
       is-index-control-unit-of-loop-statement ;

    => evaluation-effect-of(unit) #IF ($unit$)
       is-operation-or-primitive-operand #.

#DF executable-unit-successor-of (unit)

    "{unit #EQ current-executable-unit}"

    => assignment-statement-unit-successor-of (unit) #IF
       unit #IS <assignment-statement> ;

    => exchange-statement-unit-successor-of (unit) #IF unit
       #IS <exchange-statement>;
```

```
    => alternative-statement-unit-successor-of (unit) #IF
       ($unit$) is-unit-unique-to-alternative-statement;

    => conditional-statement-unit-successor-of (unit) #IF
       ($unit$) is-unit-unique-to-conditional-statement;

    => loop-statement-unit-successor-of (unit) #IF ($unit$)
       is-branch-control-unit-of-loop-statement;

    => sequential-control-unit-successor-of (unit) #IF unit
       #IS <sequence-designator> ;

    => first-unit-in (sequence-of-units-in-statement
       (statement-following (unit) ) ) #IF unit #IS
       <index-switch-declaration> #U
       <item-switch-declaration> ;

    => close-unit-successor-of (unit) #IF ($ unit $)
       is-unique-to-a-close ;

    => procedure-unit-successor (unit) #IF ($unit
       $)is-unique-to-a-procedure;

    => procedure-call-successor (unit) #IF unit #IS
       <procedure-call-statement> ;

    => return-statement-successor-of (unit) #IF unit #IS
       <return-statement>;

    => simple-successor-unit-of (unit) #IF ($unit$)
       is-index-control-unit-of-loop-statement ;

    => evaluation-successor-of(unit) #IF ($unit$)
       is-operation-or-primitive-operand #.


#DF sequence-of-assignment-statement-units-in (stmt)

    "{ stmt #IS <assignment-statement> }"

    => sequence-of-evaluation-units-in (left-hand-side-of
       (stmt)) #CS sequence-of-evaluation-units-in
       (right-hand-side-of (stmt)) #CS \stmt\ #.


#DF right-hand-side-of (stmt)
```

```
     "{ stmt #IS <assignment-statement> #U
     <exchange-statement> }"

     => #SEG 5 #OF stmt #.


#DF left-hand-side-of(stmt)

     "{stmt #IS <assignment-statement> #U
     <exchange-statement>}"

     => #SEG 1 #OF stmt #.


#DF assignment-statement-unit-successor-of (unit)

     "{ unit #IS <assignment-statement> #AND unit #EQ
     current-executable-unit }"

     => simple-successor-unit-of (unit) #.


#DF assignment-statement-effect-of (stmt)

     "{ stmt #IS <assignment-statement>}"

     => assign-effect
        (left-hand-side-of(stmt),right-hand-side-of (stmt))
        #.


#DF assign-effect (receiver-exp,valued-exp)

     => entry-assign(result-of(valued-exp), "to"
        receiving-variable-of (receiver-exp)) #IF
        receiving-variable-of(receiver-exp) #IS
        <entry-variable> #AND operand1-of (valued-exp) #IS
        <entry-variable>#U <zero>;

     => status-assign (result-of (valued-exp), "to"
        receiving-variable-of (receiver-exp)) #IF type
        (receiving-variable-of (receiver-exp)) #EQW 'status'
        #AND type (operand1-of (valued-exp)) #EQW 'status';

     => boolean-assign(result-of (valued-exp), "to"
        receiving-variable-of (receiver-exp)) #IF type
        (receiving-variable-of(receiver-exp)) #EQW 'boolean'
        #AND type (operand1-of (valued-exp)) #EQW 'boolean';
```

```
    => literal-assign (operand1-of(valued-exp), "to"
       receiving-variable-of (receiver-exp)) #IF type
       (receiving-variable-of (receiver-exp)) #IS-IN
       \'hollerith','transmission-code'\ #AND type
       (operand1-of (valued-exp)) #IS-IN
       \'hollerith','transmission-code','octal'\;

    => numeric-assign(operand1-of(valued-exp), "to"
       receiving-variable-of (receiver-exp)) #IF type
       (receiving-variable-of (receiver-exp)) #IS-IN
       \'floating','fixed','integer'\ #AND type
       (operand1-of (valued-exp)) #IS-IN
       \'floating','fixed','integer','zero'\ #.


#DF entry-assign (val,rec-var)

    => generalized-assign-latest-value
       (standard-reference-address-of( rec-var),"receives"
       ($val, "to" entry-size (rec-var)$)
       adjusted-in-leftmost-bits) #.


#DF entry-size (rec-var)

    "{rec-var #IS <entry-variable>}"

    => number-of-words-per-entry-in (declaration-for
       (tabular-name-of (rec-var))) * bits-per-word #.


#DF adjusted-in-leftmost-bits (val, "to" n "bits")

    => #RIGHT n #CHARACTERS-OF (($n$) zeroes #CW val) #.


#DF status-assign (val, rec-var)

    => generalized-assign-latest-value
       (standard-reference-address-of (rec-var), "receives"
       ($val, "to" size (rec-var)$)
       adjusted-in-leftmost-bits) #.


#DF size (rec-var)

    "{($rec-var$) is-a-variable}"
```

```
     => size-from-standard-reference-address
        (standard-reference-address-of (rec-var)) #.

#DF size-from-standard-reference-address (s-ref-addr)

     => nr-of-bits-in (s-ref-addr) #.

#DF boolean-assign (val, rec-var)

     => generalized-assign-latest-value
        (standard-reference-address-of (rec-var), "receives"
        val) #.

#DF literal-assign (operand, "to" rec-var)

     => generalized-assign-latest-value
        (standard-reference-address-of (rec-var), "receives"
        adjusted-literal (operand, "to" size (rec-var))) #.

#DF adjusted-literal (operand, "to" n "bits")

     => #RIGHT n #CHARACTERS-OF (latest-value (operand)) #IF
        #LENGTH (latest-value (operand)) > n;

     => padded-literal-value (operand, "to" n) #OTHERWISE #.

#DF numeric-assign (operand, rec-var)

     => generalized-assign-latest-value
        (standard-reference-address-of (rec-var), "receives"
        floating-latest-value (operand)) #IF type (rec-var)
        #EQW 'floating';

     => generalized-assign-latest-value
        (standard-reference-address-of (rec-var), "receives"
        integer-latest-value (operand)) #IF type (rec-var)
        #EQW 'integer';

     => generalized-assign-latest-value
        (standard-reference-address-of (rec-var), "receives"
        fixed-latest-value (operand, "with" attributes
        (rec-var))) #IF type (rec-var) #EQW 'fixed' #.
```

```
#DF fixed-latest-value (operand, "with" attr)

     => ($latest-value (operand), "to" attr $)
        converted-integer-to-fixed #IF type (operand) #EQW
        'integer';

     => ($latest-value (operand), "to" attr$)
        converted-fixed-to-fixed #IF type (operand) #EQW
        'fixed';

     => ($latest-value (operand), "to" attr$)
        converted-floating-to-fixed #IF type (operand) #EQW
        'floating' #.


#DF converted-integer-to-fixed (operand, "to" attr)

     => implementation-left-arithmetic-shift (latest-value
        (operand), "by" fraction-bits-from (attr)) #.


#DF converted-fixed-to-fixed (operand, "to" attr)

     => implementation-left-arithmetic-shift (latest-value
        (operand), "by" fraction-bits-from (attr) -
        fraction-bits-from (attributes (operand))) #.


#DF converted-floating-to-fixed (val, "to" attr)

     => #LEFT bits-per-word #CHARACTERS-OF
        implementation-left-arithmetic-shift
        (floating-mantissa-from (extended-precision-form-of
        (val)), "by" ($floating-exponent-from
        (extended-precision-form-of (val))$)
        converted-to-standard-form + 1 + fraction-bits-from
        (attr) - bits-in-floating-mantissa + 1) #.


#DF sequence-of-exchange-statement-units-in (stmt)

     "{ stmt #IS <exchange-statement> }"

     => sequence-of-evaluation-units-in
        (left-hand-side-of(stmt)) #CS
        sequence-of-evaluation-units-in
```

```
    (right-hand-side-of(stmt)) #CS \stmt\ #.


#DF exchange-statement-unit-successor-of (unit)

    "{ unit #IS <exchange-statement>}"

    => simple-successor-unit-of (unit)#.


#PROC-DF exchange-statement-effect-of (stmt)

    "{stmt #IS <exchange-statement>}"

    #COMPUTE! assign-effect (left-hand-side-of (stmt),
    right-hand-side-of (stmt))

    #COMPUTE! assign-effect (right-hand-side-of(stmt),
    left-hand-side-of (stmt))

    #RETURN-WITH-VALUE! #NIL #.


#DF sequence-of-index-switch-declaration-units-in(stmt)

    "{ stmt #IS <index-switch-declaration> }"

    => \stmt\ #CS index-switch-designator-units-implied-in
       (sequence-of-index-switch-points-in
       (index-switch-list-of(stmt)))#.


#DF index-switch-list-of(stmt)

    "{stmt #IS<index-switch-declaration>}"

    => #SEG 9 #OF stmt #.


#DF sequence-of-index-switch-points-in(iswlist)

    "{iswlist#IS<index-switch-list>}"

    => \optional-sequence-designator-of(iswlist)\ #IF
       ($iswlist$)has-just-one-optional-sequence-designator;

    => \optional-sequence-designator-of(iswlist)\ #CS
       sequence-of-index-switch-points-in
```

```
        (index-switch-list-of-trailing-designators-in
        (iswlist)) #OTHERWISE #.

#DF has-just-one-optional-sequence-designator(iswlist)

     "{iswlist #IS &lt;index-switch-list&gt;}"

     => #TRUE #IFF iswlist #IS #CASE 1 #OF
        &lt;index-switch-list&gt;#.

#DF optional-sequence-designator-of(iswlist)

     "{iswlist #IS &lt;index-switch-list&gt;}"

     => #SEG 1 #OF iswlist #.

#DF index-switch-list-of-trailing-designators-in(iswlist)

     "{ iswlist #IS #CASE 2 #OF &lt;index-switch-list&gt;}"

     => #SEG 5 #OF iswlist#.

#DF index-switch-designator-units-implied-in(seq)

     "{#FOR-ALL element #IN seg #IT-IS-TRUE-THAT (element
     #IS &lt;optional-sequence-designator&gt;}"

     => #NILSEQ #IF seq #EQ #NILSEQ;

     => index-switch-designator-units-implied-in
        (all-but-first-element-in (seq)) #IF
        #FIRST-ELEMENT-IN seq #EQW #NIL;

     => units-in-sequence-designator(sequence-designator-of
        (#FIRST-ELEMENT-IN seq)) #CS
        index-switch-designator-units-implied-in
        (all-but-first-element-in(seq)) #OTHERWISE #.

#DF sequence-of-item-switch-declaration-units-in (stmt)

     "{ stmt #IS &lt;item-switch-declaration&gt;}"

     => \stmt\ #CS item-switch-designator-units-implied-in
```

```
      (sequence-of-item-switch-points-in
      (item-switch-list-of(stmt)))#.

#DF item-switch-list-of(stmt)

    "{stmt #IS<item-switch-declaration>}"

    => #SEG 15 #OF stmt #.

#DF sequence-of-item-switch-points-in (iswlist)

    "{ iswlist #IS <item-switch-list>}"

    => \item-switch-case-expression-of(iswlist)\ #IF
       ($iswlist$)has-just-one-case-expression;

    => \item-switch-case-expression-of (iswlist)\ #CS
       sequence-of-item-switch-points-in
       (item-switch-list-of-trailing-cases-of (iswlist))
       #OTHERWISE#.

#DF item-switch-case-expression-of(iswlist)

    "{iswlist #IS <item-switch-list>}"

    => #SEG 1 #OF iswlist #.

#DF has-just-one-case-expression(iswlist)

    "{iswlist #IS <item-switch-list>}"

    => iswlist #IS #CASE 1 #OF <item-switch-list> #.

#DF item-switch-list-of-trailing-cases-of(iswlist)

    "{iswlist #IS #CASE 2 #OF <item-switch-list>}"

    => #SEG 5 #OF iswlist #.

#DF item-switch-designator-units-implied-in (seq)

    "{ #FOR-ALL element #IN seq #IT-IS-TRUE-THAT (element
```

```
    #IS <item-switch-case-expression>)}"

    => #NILSEQ #IF seq #EQ #NILSEQ ;

    => units-in-sequence-designator (sequence-designator-of
       (#FIRST-ELEMENT-IN seq))#CS
       item-switch-designator-units-implied-in
       (all-but-first-element-in(seq)) #OTHERWISE #.


#DF sequence-of-close-declaration-units-in (stmt)

    "{stmt #IS <close-declaration>}"

    => \stmt\ #CS sequence-of-close-body-units-in
       (close-body-of (stmt)) #.


#DF close-body-of (nx)

    "{nx #IS <close-subprogram> #U <close-declaration>}"

    => #SEG 7 #OF nx #.


#DF sequence-of-close-body-units-in (cbody)

    "{cbody #IS <close-body> #U <close-subprogram-body>}"

    => sequence-of-executable-units-in (inner-close-body-of
       (cbody)) #CS \close-terminator-of (cbody)\ #.


#DF inner-close-body-of (cbody)

    "{cbody #IS <close-body> #U <close-subprogram-body>}"

    => #SEG 3 #OF cbody #.


#DF close-terminator-of (cbody)

    "{cbody #IS <close-body> #U <close-subprogram-body>}"

    => #SEG 5 #OF cbody #.


#DF is-unique-to-a-close (unit)
```

```
     "{unit #EQ current-executable-unit}"

     => #TRUE #IFF unit #IS <close-declaration> #U
        <close-subprogram-term> #U <close-end> #.


#DF close-unit-successor-of (unit)

     "{unit #IS <close-declaration> #U
     <close-subprogram-term> #U <close-end>}"

     => simple-successor-unit-of (close-terminator-of
        (close-body-of (unit))) #IF unit #IS
        <close-declaration>;

     => simple-successor-unit-of (caller-of-close
        (close-containing (unit))) #IF unit #IS <close-end>
        #U <close-subprogram-term> #.


#DF close-containing (unit)

     "{unit #IS <close-end> #U <close-subprogram-term>}"

     => #LAST close #IN ( #SEQUENCE-OF-ANCESTORS-OF (unit) )
        #SUCH-THAT (close #IS <close-declaration> #U
        <close-subprogram>) #.


#DF caller-of-close (close)

     "{close #IS <close-declaration> #U <close-subprogram}"

     => #LATEST-VALUE (close-return-point-unique-to (close))
        #.


#DF close-return-point-unique-to (close)

     "{close #IS <close-declaration> #U <close-subprogram>}"

     => 'close-return-point $' #CW (#ORDPOSIT close #IN
        sequence-of-nodes-in (#ROOT-NODE (close))) #.


#DF is-unit-unique-to-go-to-statement (unit)
```

```
    "{ unit #EQ current-executable-unit }"

    => #TRUE #IFF ( unit #IS <index-switch-declaration> #U
       <item-switch-declaration> #U <sequence-designator> )
       #.

#DF sequence-of-go-to-statement-units-in(stmt)

    "{ stmt #IS <go-to-statement>}"

    =>
       units-in-sequence-designator(sequence-designator-of(stmt))#.

#DF sequence-designator-of(nx)

    "{ nx #IS <go-to-statement> #U
    <optional-sequence-designator> #U
    <item-switch-case-expression>}"

    =>  #SEG 3 #OF nx #IF nx #IS<go-to-statement>;

    => #SEG 1 #OF nx #IF nx
       #IS<optional-sequence-designator>;

    => #SEG 5 #OF nx #IF nx #IS
       <item-switch-case-expression>#.

#DF units-in-sequence-designator(sd)

    "{sd #IS sequence-designator}"

    => sequence-of-evaluation-units-in
       (destination-index-of(sd)) #CS\sd\ #IF
       ($sd$)has-a-destination-index;

    => \sd\ #OTHERWISE #.

#DF has-a-destination-index(sd)

    "{sd #IS <sequence-designator>}"

    => sd #IS #CASE 2 #OF <sequence-designator>#.
```

```
#DF sequential-control-unit-successor-of (sd)

    "{sd #IS <sequence-designator>}"

    => switched-on-successor-of (sd) #IF
       ($sd$)designates-a-switch ;

    => close-invocation-successor-of(sd) #IF
       ($sd$)constitutes-a-close-invocation;

    => named-statement-successor-of(sd) #IF
       ($sd$)designates-a-named-statement #.

#DF designates-a-switch (sd)

    "{sd #IS <sequence-designator>}"

    => #TRUE #IFF target-of (destination-name-of(sd)) #IS
       <item-switch-declaration> #U
       <index-switch-declaration> #.

#DF target-of(dest-name)

    "{ dest-name #IS <destination-name> #U
    <actual-input-close-parameter> #U
    <actual-output-destination-parameter> }"

    "{#ON-return: target-of #IS <statement> #U
    <close-declaration> #U <close-subprogram> #U
    <item-switch-declaration> #U
    <index-switch-declaration>}"

    => target-determined-using (category-2-declaration-for
       (dest-name) ) #.

#DF target-determined-using (decl)

    "{decl #IS <statement> #U <close-declaration> #U
    <index-switch-declaration> #U <item-switch-declaration>
    #U <program-declaration> #U
    <formal-input-close-parameter> #U
    <formal-output-destination-parameter>}"

    => decl #IF decl #IS <statement> #U <close-declaration>
       #U <close-subprogram> #U <index-switch-declaration>
```

```
    #U <item-switch-declaration>;

    => target-determined-using (latest-value-assigned-to
       (decl)) #IF decl #IS <formal-input-close-parameter>
       #U <formal-output-destination-parameter>;

    => library-close-subprogram-referred-to-by (decl) #IF
       decl #IS <program-declaration> #.


#DF latest-value-assigned-to (control-parameter)

    "{control-parameter #IS <formal-input-close-parameter>
    #U <formal-output-destination-parameter>}"

    => #LATEST-VALUE
       (unique-control-variable-associated-with
       (control-parameter)) #.


#DF unique-control-variable-associated-with (cp)

    "{cp #IS <formal-input-close-parameter> #U
    <formal-output-destination-parameter>}"

    => 'control-parameter-variable $' #CW (#ORDPOSIT cp #IN
       (sequence-of-nodes-in (#ROOT-NODE (cp)))) #.


#DF library-close-subprogram-referred-to-by (decl)

    "{decl #IS <program-declaration>}"

    => #FIRST close #IN
       sequence-of-library-close-subprograms-in
       (system-containing (decl)) #SUCH-THAT
       (name-declared-by (decl) #EQW name-of (close)) #.


#DF name-of (close)

    "{close #IS <close-subprogram>}"

    => #SEG 3 #OF close #.


#DF system-containing (nx)
```

```
    "{nx #IS #NODE}"

    => #ROOT-NODE (nx) #.

#DF sequence-of-library-close-subprograms-in (sys)

    "{sys #IS <jovial-j3-system>}"

    => #SEQUENCE-OF <close-subprogram> #IN
       optional-library-of (sys) #.

#DF optional-library-of (sys)

    "{sys #IS <jovial-j3-system>}"

    => #SEG 4 #OF sys #.

#DF destination-name-of(sd)

    "{ sd #IS <sequence-designator> }"

    => #SEG 1 #OF sd #.

#PROC-DF switched-on-successor-of(sd)

    "{ sd #IS <sequence-designator> #AND
    ($sd$)designates-a-switch}"

    #COMPUTE! #ASSIGN-LATEST-VALUE
    (default-switch-destination-unique-to
    (switch-designated-by(sd)), "receives"
    default-successor-of(sd))

    #RETURN-WITH-VALUE! switch-successor-of-designator(sd)
    #.

#DF default-switch-destination-unique-to (sw)

    "{ sw #IS <index-switch-declaration> #U
    <item-switch-declaration>}"

    => 'default-switch-destination$' #CW ( #ORDPOSIT sw #IN
       sequence-of-nodes-in(#ROOT-NODE(sw)) ) #.
```

#DF switch-designated-by(sd)

"{ sd #IS <sequence-designator> #AND (sd)designates-an-index-switch}"

=> target-of (destination-name-of(sd)) #.

#DF default-successor-of(sd)

"{ sd #IS <sequence-designator> #AND (sd)designates-a-switch}"

=> simple-successor-unit-of(sd) #IF innermost-executable-statement-containing(sd) #IS <go-to-statement>;

=> #LATEST-VALUE (default-switch-destination-unique-to (switch-containing(sd))) #IF innermost-executable-statement-containing(sd)#IS <item-switch-declaration> #U <index-switch-declaration> #.

#DF switch-containing(sd)

"{innermost-executable-statement-containing(sd) #IS <index-switch-declaration> #U <item-switch-declaration>}"

=> innermost-executable-statement-containing(sd) #.

#DF innermost-executable-statement-containing(nx)

"{nx #IS #NODE}"

=> #LAST ancestor #IN (#SEQUENCE-OF-ANCESTORS-OF(nx)) #SUCH-THAT (($ancestor$)is-an-executable-statement) #.

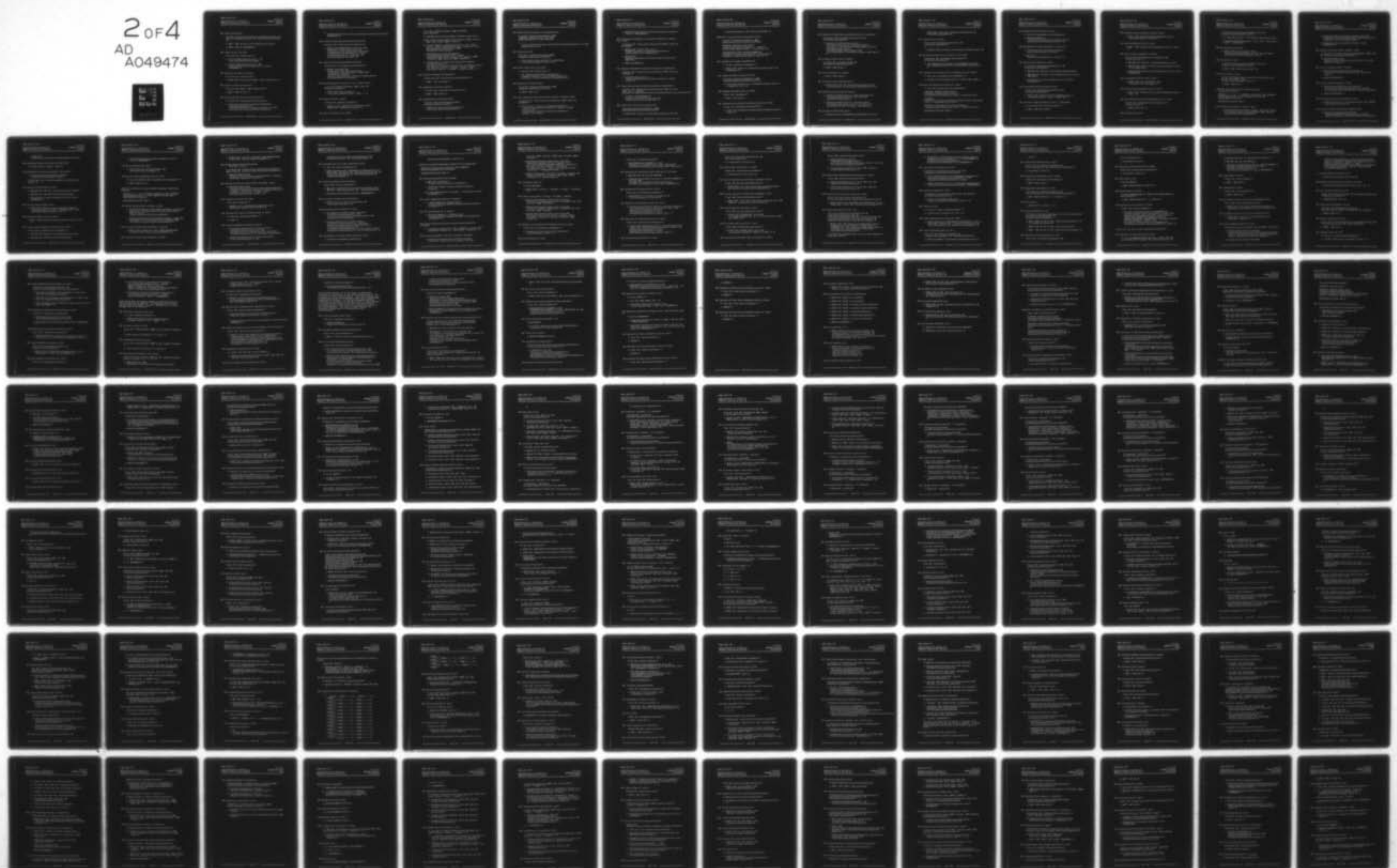
#DF switch-successor-of-designator(sd)

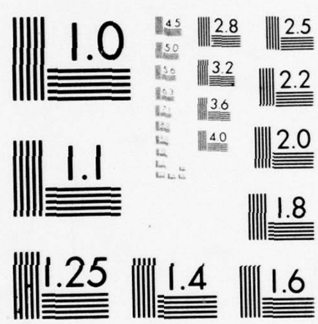
"{sd #IS <sequence-designator> #AND(sd)designates-a-switch}"

```
     => item-switch-successor-of-designator(sd) #IF
        ($sd$)designates-an-item-switch;

     => index-switch-successor-of-designator(sd) #IF
        ($sd$)designates-an-index-switch #.


#DF designates-an-item-switch(sd)

     "{ sd #IS <sequence-designator> #AND
     ($sd$)designates-a-switch}"

     => #TRUE #IFF
        switch-designated-by(sd)#IS<item-switch-declaration>#.


#PROC-DF item-switch-successor-of-designator(sd)

     "{sd #IS <sequence-designator> #AND
     ($sd$)designates-a-switch}"

     #COMPUTE! assign-latest-value (comparator-variable
     (switch-designated-by(sd)),"receives"
     comparison-variable-value-of
     (comparator-variable(switch-designated-by(sd)),"using"
     index-counts (destination-index-of(sd))))

     #RETURN-WITH-VALUE!
     unit-selected-from-item-switch(switch-designated-by
     (sd))#.


#DF comparison-variable-value-of (var, "using" index-vals)

     "{var #IS <simple-variable> #AND #FOR-ALL x #IN
     index-vals #IT-IS-TRUE-THAT ( x >= 0 ) }"

     => generalized-latest-value
        (indexed-standard-reference-address (var, "using"
        index-vals)) #.


#DF index-counts(di)

     "{di #IS <destination-index>}"

     => index-values (index-list-of (di)) #.
```

AD-A049 474 TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CALIF F/G 9/2
SEMANOL (76) SPECIFICATION OF JOVIAL (J3). VOLUME III.(U)
NOV 77 F C BELZ, I M GREEN F30602-76-C-0238
UNCLASSIFIED RADC-TR-77-365-VOL-3 NL

2 OF 4
AD
A049474





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
#DF index-list-of(nx)

    "{nx #IS <destination-index> #U <indexed-variable> #U
    <special-integer-variable> #U <special-fixed-variable>
    }"

    => #SEG 1 #OF nx #IF nx #IS <destination-index>;

    => #SEG 5 #OF nx #OTHERWISE #.


#DF index-values (ix-list)

    "{ix-list #IS <index-list>}"

    => \first-index-value(ix-list) \ #IF
       ($ix-list$)has-only-one-index;

    => \first-index-value(ix-list)\ #CS
       index-values(rest-of-index-list (ix-list))
       #OTHERWISE #.


#DF has-only-one-index (ix-list)

    "{ix-list #IS <index-list>}"

    => #TRUE #IFF ix-list #IS #CASE 1 #OF <index-list> #.


#DF rest-of-index-list(ix-list)

    "{ix-list #IS #CASE 2 #OF <index-list>}"

    => #SEG 5 #OF ix-list #.


#DF unit-selected-from-item-switch(switch)

    "{ switch #IS <item-switch-declaration>}"

    => first-unit-in(units-in-sequence-designator
       (sequence-designator-of
       (item-switch-point-selected-in (switch) ))) #IF
       a-switch-point-can-be-selected-in(switch);
```

```
    =>
       #LATEST-VALUE(default-switch-destination-unique-to(switch))
       #OTHERWISE #.


#DF a-switch-point-can-be-selected-in(switch)

    "{switch #IS <item-switch-declaration>}"

    => #TRUE #IFF #THERE-EXISTS case-expr #IN
       sequence-of-item-switch-points-in
       (item-switch-list-of(switch)) #SUCH-THAT
       (($comparator-variable(switch), "and"
       selector-constant-of (case-expr) $)
       are-determined-to-be-equal) #.


#DF item-switch-point-selected-in(switch)

    "{switch #IS <item-switch-declaration>}"

    => #FIRST case-expr #IN
       sequence-of-item-switch-points-in
       (item-switch-list-of(switch)) #SUCH-THAT (
       ($comparator-variable
       (switch),"and"selector-constant-of(case-expr)$)
       are-determined-to-be-equal)#.


#DF are-determined-to-be-equal (var, "with" const)

    "{ var #IS <simple-variable> #AND const #IS
    <general-constant>}"

    => #TRUE #IFF switch-relation
       (var,typed-constant(const)) #EQW '=' #.


#DF typed-constant (const)

    "{const #IS <general-constant>}"

    => #FIRST x #IN (#SEQUENCE-OF-NODES-IN const)
       #SUCH-THAT (($x$)is-a-constant) #.


#DF switch-relation (var,const)
```

```
        "{var #IS <simple-variable> #AND ($const$)
        is-a-constant}"

        => implementation-status-compare (latest-value (var),

        "with" latest-value(const)) #IF type(var) #EQW 'status'
           #AND type (const) #EQW 'status';

        => literal-compare( normalized-literal ( var, "wrt"
           const), "with" normalized-literal (const,"wrt" var))
           #IF ($var$)is-literal-object #AND
           ($var$)is-literal-object;

        => implementation-floating-compare
           (floating-latest-value (var), "with"
           floating-latest-value (const)) #IF type(var) #EQW
           'floating' #AND type(var) #EQW 'floating';

        => implementation-integer-and-fixed-point-compare
           (latest-value(var),
           attributes(var),latest-value(const),attributes(const))
           #IF type(var) #IS-IN \'integer','fixed'\ #AND
           type(const) #IS-IN \'integer', 'fixed','octal'\ #.


#DF selector-constant-of(case-expr)

        "{case-expr #IS <item-switch-case-expression>}"

        => #SEG 1 #OF case-expr #.


#DF comparator-variable (switch)

        "{ switch #IS <item-switch-declaration>}"

        => #SEG 7 #OF switch #.


#DF designates-an-index-switch(sd)

        "{sd #IS <sequence-designator>#AND
        ($sd$)designates-a-switch}"

        => #TRUE #IFF switch-designated-by(sd)
           #IS<index-switch-declaration>#.
```

```
#DF index-switch-successor-of-designator(sd)

     "{ sd #IS <sequence-designator> #AND
     ($sd$)has-a-destination-index #AND
     ($sd$)designates-an-index-switch}"

     =>
        unit-selected-from-index-switch(switch-designated-by(sd),"using"
        index-count(sd)) #.


#DF index-count(sd)

     "{sd #IS <sequence-designator>#AND
     ($sd$)has-a-destination-index}"

     => first-index-value (index-list-comprising
        (destination-index-of (sd) )) #.


#DF first-index-value (ix-list)

     "{ ix-list #IS <index-list> }"

     => ($ integer-latest-value (operand1-of
        (numeric-formula-of (first-index-formula-in
        (ix-list) ))) $) converted-to-standard-form #.


#DF destination-index-of(sd)

     "{ sd #IS <sequence-designator> #AND
     ($sd$)has-a-destination-index}"

     => #SEG 5 #OF sd #.


#DF unit-selected-from-index-switch (switch, "using" count)

     "{ switch #IS <index-switch-declaration> #AND count #IS
     #INTEGER}"

     => first-unit-in (units-in-sequence-designator
        (sequence-designator-selected-in (switch, "using"
        count))) #IF
        a-sequence-designator-is-properly-indexed-in
        (switch, "by" count) ;
```

```
    => #LATEST-VALUE (default-switch-destination-unique-to
       (switch)) #OTHERWISE #.

#DF a-sequence-designator-is-properly-indexed-in (switch,
"by" count)

    "{ switch #IS <index-switch-declaration>#AND count #IS
    #INTEGER}"

    => #FALSE #IF 0>count #OR count >=
       #LENGTH(sequence-of-index-switch-points-in
       (index-switch-list-of(switch)));

    => #FALSE #IF
       sequence-designator-selected-in(switch,"by"count)
       #EQW #NIL ;

    => #TRUE #OTHERWISE #.

#DF sequence-designator-selected-in(switch,"using"count)

    "{switch #IS <index-switch-declaration> #AND count #IS
    #INTEGER }"

    => sequence-designator-of
       (index-switch-point-selected-in (switch, "using"
       count)) #.

#DF index-switch-point-selected-in(switch,"using"count)

    "{switch #IS <index-switch-declaration> #AND 0<=count
    #AND count < #LENGTH (
    sequence-of-index-switch-points-in(index-switch-list-of(switch)))}

    => (count+1) #TH-ELEMENT-IN
       sequence-of-index-switch-points-in
       (index-switch-list-of(switch)) #.

#DF constitutes-a-close-invocation(sd)

    "{sd #IS <sequence-designator> #AND
    #NOT($sd$)has-a-destination-index}"

    => #TRUE #IFF target-of (destination-name-of(sd)) #IS
```

```
      <close-declaration> #U <close-subprogram> #.

#PROC-DF close-invocation-successor-of(sd)

     "{sd #IS <sequence-designator> #AND
     ($sd$)constitutes-a-close-invocation}"

     #COMPUTE! #ASSIGN-LATEST-VALUE
     (close-return-point-unique-to ( target-of
     (destination-name-of (sd)) ) ,"receives" sd)

     #RETURN-WITH-VALUE! #FIRST-ELEMENT-IN
     (sequence-of-close-body-units-in (close-body-of
     (target-of (destination-name-of (sd))))) #.

#DF designates-a-named-statement(sd)

     "{ sd #IS <sequence-designator>}"

     => #TRUE #IFF target-of (destination-name-of(sd)) #IS
        <statement> #.

#DF named-statement-successor-of(sd)

     "{sd #IS <sequence-designator> #AND
     ($sd$)designates-a-named-statement}"

     => first-executable-unit-in (unnamed-statement-part-of
        ( target-of (sd))) #.

#DF unnamed-statement-part-of (stmt)

     "{stmt #IS <statement>}"

     => #SEG 2 #OF stmt #.

#DF sequence-of-procedure-declaration-units-in (stmt)

     "{stmt #IS <procedure-declaration>}"

     => \stmt\ #CS sequence-of-executable-procedure-units-in
        (stmt) #.
```

```
#DF sequence-of-executable-procedure-units-in (proc-dec)

     "{proc-dec #IS <procedure-declaration> #U
     <procedure-subprogram>}"

     => sequence-of-executable-units-in
        (optional-decl-list-of (procedure-head-of
        (proc-dec))) #CS sequence-of-executable-units-in
        (procedure-stmt-list-of
        (procedure-body-of(proc-dec))) #CS
        \proc-terminator-of (procedure-body-of (proc-dec))\
        #.


#DF procedure-stmt-list-of (pbody)

     "{ pbody #IS <procedure-body> #U
     <procedure-subprogram-body>}"

     => #SEG 3 #OF pbody #.


#DF proc-terminator-of (pbody)

     => #SEG 5 #OF pbody #.

#DF is-unique-to-a-procedure(unit)

     => #TRUE #IFF unit #IS <procedure-declaration> #U
        <procedure-end> #U <procedure-subprogram-term> #.

#DF procedure-unit-successor (unit)

     => simple-successor-unit-of ( proc-terminator-of
        (procedure-body-of(unit))) #IF unit #IS
        <procedure-declaration>;

     => simple-successor-unit-of ( caller-of-proc
        (procedure-containing (unit))) #IF unit #IS
        <procedure-end> #U <procedure-subprogram> #.

#DF procedure-containing (unit)

     => #LAST proc #IN (#SEQUENCE-OF-ANCESTORS-OF (unit))
```

```
    #SUCH-THAT (proc #IS <procedure-declaration> #U
    <procedure-subprogram>)#.


#DF caller-of-proc (proc)

    "{proc #IS <procedure-declaration> #U
    <procedure-subprogram>}"

    =>
       #LATEST-VALUE(procedure-return-point-unique-to(proc))#.


#DF procedure-return-point-unique-to (proc-dec)

    "{proc-dec #IS <procedure-declaration> #U
    <procedure-subprogram>}"

    => 'procedure-return-point$' #CW (#ORDPOSIT proc-dec
       #IN (#SEQUENCE-OF-NODES-IN (#ROOT-NODE (proc-dec))))
       #.


#DF sequence-of-procedure-call-statement-units-in (stmt)

    "{ stmt #IS <procedure-call-statement> }"

    => sequence-of-evaluation-units-in (stmt) #CS \stmt\ #.


#PROC-DF procedure-call-effect-of (unit)

    "{ unit #IS <procedure-call-statement>}"

    #COMPUTE! #ASSIGN-LATEST-VALUE
    (procedure-return-point-unique-to
    (procedure-decl-invoked-by (unit)),"receives" unit)

    #COMPUTE!
    assignment-to-input-parameters(procedure-decl-invoked-by
    (unit) , "arguments-from" unit)

    #COMPUTE! assignment-to-output-parameters
    (procedure-decl-invoked-by (unit), "locs-from" unit)

    #RETURN-WITH-VALUE! #NIL#.
```

```
#DF procedure-call-successor (unit)

    "{ unit #IS <procedure-call-statement>}"

    => #FIRST-ELEMENT-IN
       sequence-of-executable-procedure-units-in
       (procedure-decl-invoked-by (unit)) #.


#DF assignment-to-input-parameters (proc,call)

    => successive-input-parameter-assigns (
       seq-of-input-params-in (proc),

    "receiving"
       seq-of-input-arguments-in (call)) #.


#DF seq-of-input-params-in (proc)

    "{ proc #IS <procedure-declaration > #U
    <procedure-subprogram>}"

    => #NILSEQ #IF ($proc$) is-parameterless-procedure;

    => #NILSEQ #IF optional-input-parameter-list-of (proc)
       #EQW #NIL;

    => arg-seq-from (input-parameter-list-of (proc))
       #OTHERWISE #.


#DF is-parameterless-procedure (proc)

    "{ proc #IS <procedure-declaration> #U
    <procedure-subprogram>}"

    => #TRUE #IFF optional-formal-parameter-list-of
       (procedure-head-of (proc) ) #EQW #NIL #.


#DF optional-formal-parameter-list-of (proc-head)

    "{ proc-head #IS <procedure-head> }"

    => #SEG 5 #OF proc-head #.
```

```
#DF optional-input-parameter-list-of (proc)

    => #SEG 3 #OF optional-formal-parameter-list-of
       (procedure-head-of (proc)) #.

#DF input-parameter-list-of (proc)

    => #SEG 1 #OF optional-input-parameter-list-of (proc)
       #.

#DF seq-of-input-arguments-in (call)

    "{ call #IS <procedure-call-statement> #U
    <function-call> }"

    => #NILSEQ #IF ($call$) is-argumentless-proc-call;

    => #NILSEQ #IF optional-actual-input-parameter-list-of
       (call) #EQW #NIL;

    => arg-seq-from (actual-input-parameter-list-of (call))
       #OTHERWISE#.

#DF is-argumentless-proc-call (call)

    => #TRUE #IFF call #IS #CASE 1 #OF
       <procedure-call-statement> #.

#DF actual-input-parameter-list-of (call)

    => #SEG 1 #OF optional-actual-input-parameter-list-of
       (call) #.

#DF optional-actual-input-parameter-list-of (call)

    "{call #IS <procedure-call-statement> #U
    <function-call>}"

    => #SEG 4 #OF call #.

#DF arg-seq-from (apl)
```

```
     "{ apl #IS <actual-input-parameter-list> #U
     <actual-output-parameter-list>}"

     => \ last-arg-of (apl) \ #IF ($apl$) has-only-one-arg ;

     => arg-seq-from (first-args-of (apl)) #CS \ last-arg-of
        (apl) \ #OTHERWISE #.


#DF has-only-one-arg (apl)

     => #TRUE #IFF apl #IS #CASE 1 #OF
        <actual-input-parameter-list> #OR apl #IS #CASE 1
        #OF <actual-output-parameter-list> #.


#DF last-arg-of (apl)

     "{ apl #IS <actual-input-parameter-list> #U
     <actual-output-parameter-list>}"

     => #SEG 1 #OF (last-seg-of (apl)) #.


#DF first-args-of (apl)

     "{ apl #IS #CASE 2 #OF <actual-input-parameter-list>
     #OR apl #IS #CASE 2 #OF
     <actual-output-parameter-list>}"

     => #SEG 1 #OF apl #.


#PROC-DF successive-input-parameter-assigns
  (param-seq, arg-seq)

  #FOR-ALL i : 1 <= i <= #LENGTH (param-seq) #DO #COMPUTE!
  assign-input-param (i #TH-ELEMENT-IN param-seq, i
  #TH-ELEMENT-IN arg-seq)

  #RETURN-WITH-VALUE! #NIL #.


#DF assign-input-param ( iparam, iarg )

     => assign-close-input-param (iparam, iarg) #IF iparam
        #IS <formal-input-close-parameter> #AND iarg #IS
        <actual-input-close-parameter>;
```

```
     => assign-by-name-input-param (iparam, iarg) #IF
        ($iparam$) is-call-by-name-formal #AND ($operand1-of
        (iarg)$) is-call-by-name-argument;

     => assign-by-value-input-param (iparam, iarg)
        #OTHERWISE #.


#DF assign-close-input-param (iparam, iarg)

     "{ iparam #IS <formal-input-close-parameter> #AND iarg
     #IS <actual-input-close-parameter>}"

     => #ASSIGN-LATEST-VALUE
        (unique-control-variable-associated-with (iparam),
        target-of (iarg)) #.


#DF is-call-by-name-formal (iparam)

     => ($detailed-declaration-for (iparam) $)
        is-a-table-or-array-declaration #IF iparam #IS
        <name>;

     => #FALSE #OTHERWISE #.


#DF is-call-by-name-argument (iarg)

     => ($detailed-declaration-for (iarg) $)
        is-a-table-or-array-declaration #IF ($iarg$)
        is-a-variable #AND iarg #IS-NOT <indexed-variable>;

     => #FALSE #OTHERWISE #.


#DF is-a-table-or-array-declaration (dec)

     => #TRUE #IFF dec #IS <array-declaration> #OR ($dec$)
        is-a-table-declaration #.


#DF assign-by-name-input-param (iparam,iarg)

     => generalized-assign-latest-value
        (simple-standard-reference-address (iparam),
        ($word-address-from (standard-reference-address-of
```

```
      (iarg)) $)
      with-result-converted-to-implementation-form) #.

#DF assign-by-value-input-param (iparam,iarg)

    => assign-effect (iparam, iarg) #.

#DF assignment-to-output-parameters (proc,call)

    => successive-output-parameter-assigns
       (seq-of-output-params-in (proc),
    "receiving" seq-of-output-argument-locations-from
    (call)) #.

#DF seq-of-output-params-in (proc)

    => #NILSEQ #IF ($proc$) is-parameterless-procedure;

    => #NILSEQ #IF ($proc$) has-no-output-params;

    => arg-seq-from (output-parameter-list-of (proc))
       #OTHERWISE #.

#DF has-no-output-params (proc)

    => #TRUE #IFF optional-formal-parameter-list-of
       (procedure-head-of (proc) ) #IS-NOT #CASE 3 #OF
       <optional-formal-parameter-list> #.

#DF output-parameter-list-of (proc)

    => #SEG 7 #OF optional-formal-parameter-list-of (proc)
       #.

#DF seq-of-output-argument-locations-from (call)

    "{call #IS <procedure-call-statement>}"

    => #NILSEQ #IF ($call$) is-argumentless-proc-call;

    => #NILSEQ #IF ($call$) has-no-output-args;
```

```
    => arg-seq-from (actual-output-parameter-list-of
       (call)) #OTHERWISE #.


#DF has-no-output-args (proc)

    => #TRUE #IFF proc #IS-NOT #CASE 3 #OF
       <procedure-call-statement> #.


#DF actual-output-parameter-list-of (call)

    "{ call #IS #CASE 3 #OF <procedure-call-statement> }"

    => #SEG 9 #OF call #.


#PROC-DF successive-output-parameter-assigns (param-seq,
arg-seq)

  #FOR-ALL i : 1 <=i<= #LENGTH (param-seq) #DO #COMPUTE!
  assign-output-param ( i #TH-ELEMENT-IN param-seq, i
  #TH-ELEMENT-IN arg-seq)

  #RETURN-WITH-VALUE! #NIL #.


#DF assign-output-param (o-param, o-arg)

    => assign-destination-output-param (o-param, o-arg) #IF
       o-param #IS <formal-output-destination-parameter>
       #AND o-arg #IS
       <actual-output-destination-parameter>;

    => assign-ref-addr-to-output-param (o-param, o-arg) #IF
       ($o-param$) is-formal-output-data-parameter #AND
       ($o-arg$) is-actual-output-data-parameter #.


#DF is-formal-output-data-parameter (o-param)

    => #TRUE #IFF o-param #IS <name> #AND #PARENT-NODE
       (o-param) #IS <formal-output-parameter> #.


#DF is-actual-output-data-parameter (o-arg)
```

```
    => #TRUE #IFF o-arg #IS <variable> #AND #PARENT-NODE
       (o-arg) #IS <actual-output-parameter> #.


#DF assign-destination-output-param
       (o-param, o-arg)

    "{ o-param #IS <formal-output-destination-parameter>
    #AND o-arg #IS <actual-output-destination-parameter>}"

    => #ASSIGN-LATEST-VALUE
       (unique-control-variable-associated-with (o-param),
       "receives" target-of (o-arg)) #.


#DF assign-ref-addr-to-output-param (o-param, o-arg)

    => #ASSIGN-LATEST-VALUE
       (unique-reference-address-variable-associated-with
       (o-param), "receives" \standard-reference-address-of
       (variable-constituting (o-arg)), "and"
       variable-constituting (o-arg) \) #.


#DF variable-constituting (var)

    "{var #IS <variable> }"

    => #FIRST nx #IN (#SEQUENCE-OF-NODES-IN (var))
       #SUCH-THAT (($nx$) is-a-variable) #.


#DF sequence-of-return-statement-units-in (stmt)

    "{ stmt #IS <return-statement> }"

    => \stmt\ #.


#DF return-statement-successor-of(unit)

    => simple-successor-unit-of (caller-of-proc
       (procedure-containing(unit))) #IF
       innermost-proc-or-close-containing (unit) #IS
       <procedure-declaration> #U <procedure-subprogram>;

    => simple-successor-unit-of (caller-of-close (
       close-containing(unit))) #IF
```

```
     innermost-proc-or-close-containing(unit) #IS
     <close-declaration> #U <close-subprogram> #.


#DF innermost-proc-or-close-containing (unit)

    "{ unit #IS <return-statement>}"

    => #LAST candidate #IN (#SEQUENCE-OF-ANCESTORS-OF unit)
       #SUCH-THAT (candidate #IS <procedure-declaration> #U
       <procedure-subprogram> #U <close-declaration> #U
       <close-subprogram>) #.


#DF return-statement-effect-of(unit)

    "{unit #IS <return-statement>}"

    => #NIL #IF innermost-proc-or-close-containing (unit)
       #IS-IN < close-declaration> #U < close-subprogram>;

    => procedure-return-effect-of(unit) #OTHERWISE #.


#DF is-a-return-from-a-procedure(unit)

    => #TRUE #IFF unit #IS <procedure-end> #U
       <procedure-subprogram> #.


#DF procedure-return-effect-of (unit)

    => assignment-to-actual-output-args-from
       (procedure-containing(unit)) #IF
       caller-of-proc(procedure-containing(unit))#IS
       <procedure-call-statement>;

    => assign-latest-value (caller-of-proc
       (procedure-containing(unit)),"receives"
       generalized-latest-value (name-declared-in
       (function-item-declaration-for
       (procedure-containing(unit))))) #IF caller-of-proc
       (procedure-containing(unit)) #IS <function-call> #.


#DF assignment-to-actual-output-args-from (proc)

    => successive-argument-assigns-from
```

```
    (seq-of-output-params-in (proc)) #.


#PROC-DF successive-argument-assigns-from (o-param-seq)

  #FOR-ALL o-param #IN o-param-seq #DO #COMPUTE!
  output-arg-assign-from (o-param)

  #RETURN-WITH-VALUE! #NIL #.


#DF output-arg-assign-from (o-param)

     => #NIL #IF o-param #IS
        <formal-output-destination-parameter>;

     => output-arg-assign (output-argument-passed-to
        (o-param),

     "at" ref-addr-passed-to (o-param) , "receives" o-param)
        #OTHERWISE #.


#DF output-argument-passed-to (o-param)

     => 2 #TH-ELEMENT-IN( #LATEST-VALUE
        (unique-reference-address-variable-associated-with
        (o-param))) #.


#DF ref-addr-passed-to (o-param)

     => #FIRST-ELEMENT-IN ( #LATEST-VALUE
        (unique-reference-address-variable-associated-with
        (o-param))) #.


#DF unique-reference-address-variable-associated-with
(o-param)

     => 'output-var-addr-loc$' #CW ( #ORDPOSIT o-param #IN (
        #SEQUENCE-OF-NODES-IN ( #ROOT-NODE (o-param)))) #.


#DF output-arg-assign (rec-var, rec-addr, o-param)

     => generalized-assign-latest-value (rec-addr,
        "receives" latest-value (o-param)) #IF type
```

```
        (rec-var) #EQW 'boolean' #AND type (o-param) #EQW
        'boolean';

    => generalized-assign-latest-value (rec-addr,
        "receives" adjusted-literal (o-param, "to"
        size-from-standard-reference-address (rec-addr)))
        #IF type (rec-var) #IS-IN \ 'hollerith',
        'transmission-code'\ #AND type (o-param) #IS-IN \
        'hollerith', 'transmission-code' \;

    => numeric-arg-assign (rec-var, rec-addr, o-param) #IF
        ($type (rec-var)$) is-numeric-type #AND ($type
        (o-param) $) is-numeric-type #.


#DF is-numeric-type (t)

    "{t #IS #STRING}"

    => #TRUE #IFF t #IS-IN \ 'integer', 'fixed', 'floating'
        \ #.


#DF numeric-arg-assign (rec-var, rec-addr, o-param)

    => generalized-assign-latest-value (rec-addr,
        "receives" floating-latest-value (o-param)) #IF type
        (rec-var) #EQW 'floating';

    => generalized-assign-latest-value (rec-addr,
        "receives" integer-latest-value (o-param)) #IF type
        (rec-var) #EQW 'integer';

    => generalized-assign-latest-value (rec-addr,
        "receives" fixed-latest-value (o-param, "with"
        attributes (rec-var))) #IF type (rec-var) #EQW
        'fixed' #.

#DF sequence-of-alternative-statement-units-in (stmt)

    "{stmt #IS <alternative-statement>}"

    => sequence-of-alternative-units-implied-in
        (alternatives-of (stmt)) #.


#DF alternatives-of (stmt)
```

```
    "{stmt #IS <alternative-stmt>}"

    => #SUBSEQUENCE-OF-ELEMENTS alt #IN
       sequence-of-alternatives-of (stmt) #SUCH-THAT
       (alternative-statement-containing (alt) #EQ stmt) #.


#DF sequence-of-alternative-units-implied-in (alt-seq)

    => #NILSEQ #IF alt-seq #EQ #NILSEQ;

    => sequence-of-alternative-units-in (#FIRST-ELEMENT-IN
       alt-seq) #CS
       sequence-of-alternative-units-implied-in(
       all-but-first-element-in (alt-seq)) #OTHERWISE #.


#DF sequence-of-alternatives-of (stmt)

    "{stmt #IS <alternative-statement>}"

    => #SEQUENCE-OF <if-either-alternative> #U
       <or-if-alternative> #IN stmt #.


#DF sequence-of-alternative-units-in (alt)

    => sequence-of-evaluation-units-in (boolean-formula-of
       (alt)) #CS\alternative-test-point-of (alt)\ #CS
       sequence-of-executable-units-in
       (independent-statement-of (alt))
       #CS\unwritten-alternative-end-of (alt)\ #.


#DF sequence-of-conditional-units-in (stmt)

    "{stmt #IS <conditional-statement>}"

    => sequence-of-evaluation-units-in (boolean-formula-of
       (stmt)) #CS \conditional-test-point-of (stmt)\ #CS
       sequence-of-executable-units-in
       (independent-statement-of (stmt))
       #CS\unwritten-conditional-end-of (stmt)\ #.


#DF alternative-test-point-of (alt)
```

```
    "{alt #IS <if-either-alternative> #U
    <or-if-alternative>}"

    => reverse-seg (4,"of"alt) #.

#DF conditional-test-point-of (stmt)

    "{stmt #IS <conditional-statement>}"

    => reverse-seg (4,"of" stmt) #.

#DF is-unit-unique-to-alternative-statement (unit)

    "{unit #EQ current-executable-unit}"

    => #TRUE #IFF (unit #IS <unwritten-alternative-end>)
       #OR ($unit$) is-alternative-test-point #.

#DF is-unit-unique-to-conditional-statement (unit)

    "{unit #EQ current-executable-unit}"

    => #TRUE #IFF (unit #IS <unwritten-conditional-end> #OR
       ($unit$) is-conditional-test-point) #.

#DF is-alternative-test-point (unit)

    "{unit #EQ current-executable-unit}"

    => #TRUE #IFF #PARENT-NODE (unit) #IS
       <if-either-alternative> #U <or-if-alternative> #AND
       unit #EQW '$'#.

#DF is-conditional-test-point (unit)

    "{unit #EQ current-executable-unit}"

    => #TRUE #IFF #PARENT-NODE (unit) #IS
       <conditional-statement> #AND unit #EQW '$' #.

#DF alternative-statement-unit-successor-of (unit)
```

```
    "{unit #EQ current-executable-unit}"

    => simple-successor-unit-of
       (alternative-statement-end-of
       (alternative-statement-containing (unit))) #IF unit
       #IS <unwritten-alternative-end> ;

    => conditional-successor-of (unit) #IF ($unit$)
       is-alternative-test-point #.


#DF conditional-statement-unit-successor-of (unit)

    "{unit #EQ current-executable-unit}"

    => simple-successor-unit-of (unit) #IF unit #IS
       <unwritten-conditional-end> ;

    => conditional-successor-of (unit) #IF ($unit$)
       is-conditional-test-point #.


#DF alternative-statement-containing (unit)

    "{unit #IS <if-either-alternative> #U
    <or-if-alternative> #U <unwritten-alternative-end>}"

    => #LAST altern #IN (#SEQUENCE-OF-ANCESTORS-OF (unit) )
       #SUCH-THAT (altern #IS <alternative-statement>) #.


#DF simple-successor-unit-of (nx)

    "{nx #IS <alternative-statement-end> #U
    <unwritten-alternative-end> #U
    <unwritten-conditional-end> #U <initial-formula> #U
    <increment-formula> #U <termination-formula> #U
    <special-test-statement> #U <go-to-statement>}"

    => #FIRST unit #IN sequence-of-executable-units-in
       (program-unit-containing(nx)) #SUCH-THAT (nx
       #PRECEDES unit #IN sequence-of-executable-units-in
       (program-unit-containing(nx))) #IF nx #IS
       <alternative-statement-end>;

    "<alternative-statement-end> is not in the sequence of
    executable units."
```

```
     => (#ORDPOSIT nx #IN sequence-of-executable-units-in
        (program-unit-containing(nx))) + 1 #TH-ELEMENT-IN
        sequence-of-executable-units-in
        (program-unit-containing (nx)) #OTHERWISE #.

#DF conditional-successor-of (unit)

     "{($unit$) is-alternative-test-point #OR
     ($unit$)is-conditional-test-point}"

     => first-executable-unit-in(independent-statement-of
        (conditional-phrase-containing(unit))) #IF
        ($condition-tested-by(unit)$) is-true;

     => simple-successor-unit-of (conditional-phrase-end-of
        (conditional-phrase-containing(unit))) #OTHERWISE #.

#DF condition-tested-by (unit)

     "{($unit$) is-alternative-test-point #OR ($unit$)
     is-conditional-test-point}"

     => result-of (boolean-formula-of
        (conditional-phrase-containing (unit))) #.

#DF result-of (nx)

     "{ ($ nx $) is-an-evaluated-formula }"

     => latest-value (operand1-of (nx) ) #.

#DF conditional-phrase-containing (unit)

     => #LAST phrase #IN (#SEQUENCE-OF-ANCESTORS-OF (unit) )
        #SUCH-THAT (phrase #IS <if-either-alternative> #U
        <or-if-alternative> #U <conditional-statement>) #.

#DF first-executable-unit-in (nx)

     "{nx #IS <alternative-statement> #U
     <conditional-statement> #U <unnamed-statement>}"

     => #FIRST-ELEMENT-IN sequence-of-executable-units-in
```

```
     (nx) #.

#DF conditional-phrase-end-of (unit)

     "{unit #IS <if-either-alternative> #U
     <or-if-alternative> #U <conditional-statement>}"

     => last-seg-of (unit) #.

#DF alternative-statement-end-of (stmt)

     "{stmt #IS <alternative-statement>}"

     => #SEG 3 #OF stmt #.

#DF independent-statement-of (alt)

     "{alt #IS <if-either-alternative> #U
     <or-if-alternative> #U <conditional-statement>}"

     => #SEG (#SEG-COUNT(alt) - 1) #OF alt #.

#DF is-true(val)

     => #TRUE #IFF val #EQW '1' #.

#DF boolean-formula-of (nx)

     "{nx #IS <if-either-alternative> #U <or-if-alternative>
     #U <conditional-statement> #U
     <special-test-statement>}"

     => #SEG 3 #OF nx #IF nx #IS <conditional-statement> #U
     <if-either-alternative>;

     => #SEG 4 #OF nx #IF nx #IS <or-if-alternative>;

     => #SEG 1 #OF nx #IF nx #IS <special-test-statement> #.

#DF unwritten-alternative-end-of (alt)

     "{alt #IS <if-either-alternative> #U
```

```
     <or-if-alternative>}"

     => last-seg-of (alt) #.


#DF unwritten-conditional-end-of (stmt)

     "{stmt #IS <conditional-statement>}"

     => last-seg-of (stmt) #.


#DF last-seg-of (nx)

     "{nx #IS #NODE}"

     => #SEG (#SEG-COUNT(nx)) #OF nx #.


#DF reverse-seg (n,"of"nx)

     "{n #IS #INTEGER #AND n>0 #AND nx #IS #NODE #AND n<=
     #SEG-COUNT(nx)}"

     => #SEG (#SEG-COUNT(nx)+1 - n) #OF nx #.


#DF sequence-of-loop-units-in (stmt)

     "{stmt #IS <loop-statement>}"

     => sequence-of-formula-units-in (#SEQUENCE-OF
        <initial-formula> #IN loop-header-of (stmt)) #CS
        sequence-of-executable-units-in (loop-body-of
        (stmt)) #CS sequence-of-formula-units-in
        (#REVERSE-SEQUENCE (#SEQUENCE-OF <increment-formula>
        #IN loop-header-of (stmt))) #CS
        sequence-of-formula-units-in (#SEQUENCE-OF
        <termination-formula> #IN loop-header-of (stmt)) #.


"There will be one or zero termination-formulas."


#DF sequence-of-formula-units-in (seq)

     "{ seq #IS #SEQUENCE #AND #FOR-ALL element #IN seq
     #IT-IS-TRUE-THAT (element #IS <initial-formula> #U
```

```
    <increment-formula> #U <termination-formula>) }"

    => #NILSEQ #IF seq #EQ #NILSEQ;

    => sequence-of-evaluation-units-in (#FIRST-ELEMENT-IN
       seq) #CS \#FIRST-ELEMENT-IN seq\ #CS
       sequence-of-formula-units-in
       (all-but-first-element-in (seq)) #OTHERWISE #.


#DF loop-header-of(stmt)

    "{stmt #IS <loop-statement>}"

    => #SEG 1 #OF stmt #.


#DF loop-body-of (stmt)

    "{stmt #IS <loop-statement>}"

    => #SEG 2 #OF stmt #.


#DF is-branch-control-unit-of-loop-statement(unit)

    "{unit #EQ current-executable-unit}"

    => #TRUE #IFF (unit #IS <termination-formula> #U
       <special-test-statement> #U <test-statement>) #.


#DF is-index-control-unit-of-loop-statement(unit)

    => #TRUE #IFF (unit #IS <initial-formula> #U
       <increment-formula>) #.


#DF loop-statement-effect-of (unit)

    "{unit #IS <initial-formula> #U <increment-formula>}"

    => assign-latest-value-of-variable
       (loop-control-variable-in (for-clause-containing
       (unit)), "receives" result-of (numeric-formula-of
       (unit))) #IF unit #IS <initial-formula>;

    => assign-latest-value-of-variable
```

```
        (loop-control-variable-in (for-clause-containing
        (unit)), "receives" integer-add (integer-result-of
        (numeric-formula-of (unit)) , "and"
        latest-value-of-variable (loop-control-variable-in
        (for-clause-containing (unit))))) #IF unit #IS
        <increment-formula> #.


#DF latest-value-of-variable (var)

     => generalized-latest-value (var) #.


#DF assign-latest-value-of-variable (var,val)

     => generalized-assign-latest-value (var, val) #.


#DF for-clause-containing (unit)

     "{unit #IS <initial-formula> #U <increment-formula> #U
     <termination-formula>}"

     => #PARENT-NODE (unit) #.


#DF loop-control-variable-in (nx)

     "{nx #IS <one-facter-for-clause> #U
     <two-factor-for-clause> #U <complete-for-clause>}"

     => #SEG 3 #OF nx #.


#DF numeric-formula-of (unit)

     "{unit #IS <initial-formula> #U <increment-formula> #U
     <termination-formula> #U <index-formula>}"

     => #SEG 1 #OF unit #.


#DF integer-result-of (nx)

     "{ ($ nx $) is-an-evaluated-formula }"

     => integer-latest-value (operand1-of (nx) ) #.
```

```
#DF loop-statement-unit-successor-of (unit)

     "{unit #IS <termination-formula> #U
     <special-test-statement> #U <test-statement>}"

     => termination-formula-unit-successor-of (unit) #IF
        unit #IS <termination-formula>;

     => special-test-statement-unit-successor-of (unit) #IF
        unit #IS <special-test-statement>;

     => test-statement-unit-successor-of (unit) #IF unit #IS
        <test-statement> #.

#DF termination-formula-unit-successor-of (unit)

     "{unit #IS <termination-formula>}"

     => simple-successor-unit-of (unit) #IF
        loop-range-is-satisfied-for (unit);

     => beginning-of-loop-body-containing (unit) #OTHERWISE
        #.

#DF beginning-of-loop-body-containing (unit)

     "{unit #IS <termination-formula>}"

     => #FIRST-ELEMENT-IN sequence-of-executable-units-in
        (loop-body-of (loop-statement-containing (unit))) #.

#DF loop-statement-containing (unit)

     "{unit #IS <termination-formula> #U
     <special-test-statement>}"

     => #LAST nx #IN (#SEQUENCE-OF-ANCESTORS-OF (unit) )
        #SUCH-THAT (nx #IS <loop-statement>) #.

#DF loop-range-is-satisfied-for (unit)

     "{unit #IS <termination-formula>}"
```

```
     => is-less-in-value (loop-control-variable-in
        (for-clause-containing (unit) ), "than"
        numeric-formula-of (unit) ) #IF ($
        numeric-formula-of (increment-formula-implied-by
        (unit) ) $) has-value-less-than-zero;

     => is-greater-in-value (loop-control-variable-in
        (for-clause-containing (unit) ), "than"
        numeric-formula-of (unit) ) #OTHERWISE #.

"AFM 100-24 does not define whether an increment value of
zero is positive or negative.  To determine if the loop
variable has gone beyond its limit unambiguously, we have
assumed zero to be positive."

#DF has-value-less-than-zero (fx)

     "{ fx #IS <numeric-formula> }"

     => #TRUE #IFF relation-with-integer-zero-of
        (integer-result-of (fx) ) #EQW '<' #.

#DF is-less-in-value (nx,ny)

     "{ nx #IS <loop-variable> #AND ny #IS <numeric-formula>
     }"

     => have-values-in-relation ('<',nx,ny) #.

#DF is-greater-in-value (nx,ny)

     "{ nx #IS <loop-variable> #AND ny #IS <numeric-formula>
     }"

     => have-values-in-relation ('>',nx,ny) #.

#DF have-values-in-relation (rel,nx,ny)

     "{ nx #IS <loop-variable> #AND ny #IS <numeric-formula>
     #AND rel #IS-IN \'<','>','='\ }"

     => #TRUE #IFF rel #EQW
        implementation-integer-and-fixed-point-compare
```

```
        (latest-value (nx), "with" attributes (nx), "with"
        integer-result-of (ny), "with"
        integer-attributes(ny) ) #.


#DF increment-formula-implied-by (unit)

     "{unit #IS <termination-formula>}"

     => #FIRST nx #IN sequence-of-executable-units-in
        (loop-statement-containing (unit)) #SUCH-THAT (nx
        #IS <increment-formula>) #.


#DF special-test-statement-unit-successor-of (unit)

     "{unit #IS <special-test-statement>}"

     => simple-successor-unit-of(unit) #IF($result-of
        (boolean-formula-of(unit))$) is-true;

     =>
        simple-successor-unit-of-loop-statement-containing(unit)
        #OTHERWISE #.


#DF simple-successor-unit-of-loop-statement-containing(unit)

     "{unit #IS <special-test-statement>}"

     => #FIRST ex-unit #IN sequence-of-executable-units-in
        (program-unit-containing (unit)) #SUCH-THAT
        (#LAST-ELEMENT-IN sequence-of-executable-units-in
        (loop-statement-containing(unit)) #PRECEDES ex-unit
        #IN sequence-of-executable-units-in
        (program-unit-containing(unit) )) #.


#DF sequence-of-test-units-in (stmt)

     => \stmt\ #IF stmt #IS <test-statement>;

     => sequence-of-evaluation-units-in (stmt) #IF stmt #IS
        <special-test-statement> #.


#DF test-statement-unit-successor-of(unit)
```

```
    "{unit #IS <test-statement>}"

    => first-evaluation-unit-in
       (increment-formula-referenced-by(unit) ) #.
```

"An increment-formula of a for clause is referenced (directly or indirectly) by the test-statement and is the 'location'to which control passes. The test-statement must be contained in the loop-statement which also contains the referenced for-clause. If the test-statement directly references a one-factor-for-clause of such a loop-statement, it is assumed that there is at least one syntactically preceding two-factor-for-clause or complete-for-clause in the loop-statement. The last of these contains the increment-formula which is referenced by the test-statement."

```
#DF first-evaluation-unit-in(nx)

    "{nx #IS <increment-formula>}"

    => #FIRST-ELEMENT-IN
       sequence-of-evaluation-units-in(nx) #.

#DF increment-formula-referenced-by(unit)

    "{unit #IS <test-statement>}"

    => #SEG 11 #OF for-clause-referenced-by(unit) #.

#DF for-clause-referenced-by(unit)

    "{unit #IS <test-statement>}"

    => for-clause-directly-referenced-by(unit) #IF
       for-clause-directly-referenced-by(unit) #IS
       <two-factor-for-clause> #U <complete-for-clause>;

    => #LAST nx #IN sequence-of-for-clauses-implied-in
       (sequence-of-loop-statements-containing(unit))
       #SUCH-THAT (nx #PRECEDES
       for-clause-directly-referenced-by(unit) #IN
       sequence-of-for-clauses-implied-in
       (sequence-of-loop-statements-containing(unit)) #AND
```

```
        nx #IS <two-factor-for-clause> #U
        <complete-for-clause> ) #IF
        for-clause-directly-referenced-by(unit) #IS
        <one-factor-for-clause> #.


#DF for-clause-directly-referenced-by(unit)

    "{unit #IS <test-statement>}"

    => #LAST for-clause #IN
       sequence-of-for-clauses-implied-in
       (sequence-of-loop-statements-containing(unit))
       #SUCH-THAT (loop-variable-of(for-clause) #EQW
       loop-variable-of(unit) ) #IF
       ($unit$)declares-loop-variable;

    => #LAST-ELEMENT-IN sequence-of-for-clauses-implied-in
       (sequence-of-loop-statements-containing(unit))
       #OTHERWISE #.


#DF sequence-of-for-clauses-implied-in(loop-statement-seq)

    "{loop-statement-seq #IS #SEQUENCE #AND #FOR-ALL
    element #IN loop-statement-seq #IT-IS-TRUE-THAT
    (element #IS <loop-statement>)}"

    => #NILSEQ #IF loop-statement-seq #EQ #NILSEQ;

    => #SEQUENCE-OF <one-factor-for-clause> #U
       <two-factor-for-clause> #U <complete-for-clause> #IN
       loop-header-of (#FIRST-ELEMENT-IN
       loop-statement-seq) #CS
       sequence-of-for-clauses-implied-in
       (all-but-first-element-in(loop-statement-seq) )
       #OTHERWISE #.


#DF loop-variable-of(unit)

    "{unit #IS <one-factor-for-clause> #U
    <two-factor-for-clause> #U <complete-for-clause> #U
    <test-statement>}"

    => #SEG 3 #OF unit #IF unit #IS <one-factor-for-clause>
       #U <two-factor-for-clause> #U <complete-for-clause>;
```

```
    => #SEG 3 #OF unit #IF ($unit$)declares-loop-variable
       #.

#DF declares-loop-variable(unit)

    "{unit #IS <test-statement>}"

    => #TRUE #IFF unit #IS #CASE 2 #OF <test-statement> #.

#DF sequence-of-loop-statements-containing(unit)

    "{unit #IS <test-statement>}"

    => #SUBSEQUENCE-OF-ELEMENTS nx #IN
       #SEQUENCE-OF-ANCESTORS-OF (unit) #SUCH-THAT (nx #IS
       <loop-statement> #AND nx #IS-IN
       program-unit-containing(unit)) #.

#DF is-an-executable-statement (nx)

    "{nx#IS#NODE}"

    => nx #IS-IN sequence-of-executable-statements-in
       (program-unit-containing(nx))#.

#DF first-unit-in(seq)

    => #FIRST-ELEMENT-IN(seq)#.

#DF statement-following(stmt)

    "{stmt #IS-IN sequence-of-executable-statements-in
    (current-program-containing(stmt))}"

    => ((#ORDPOSIT stmt #IN
       sequence-of-executable-statements-in
       (program-unit-containing(stmt))) + 1 )
       #TH-ELEMENT-IN sequence-of-executable-statements-in
       (program-unit-containing (stmt))#.

#DF sequence-of-evaluation-units-in(nx)
```

```
    "{ ($ nx $) is-an-evaluated-formula }"

    => #SUBSEQUENCE-OF-ELEMENTS unit #IN
       postorder-sequence-of-nodes-in (nx) #SUCH-THAT ( ($
       unit $) is-operation-or-primitive-operand ) #.


#DF postorder-sequence-of-nodes-in (nx)

    "{ nx #IS #NODE }"

    => \nx\ #IF #SEG-COUNT (nx) = 0;

    => postorder-sequence-for-segs-of (nx
       ,"starting-with-seg" 1) #CS \nx\ #OTHERWISE #.


#DF postorder-sequence-for-segs-of (nx, "starting with seg"
n)

    "{seq #IS #SEQUENCE}"

    => postorder-sequence-of-nodes-in (#SEG n #OF nx) #IF n
       = #SEG-COUNT(nx);

    => postorder-sequence-of-nodes-in (#SEG n #OF nx) #CS
       postorder-sequence-for-segs-of (nx, "continuing with
       seg" n+1) #OTHERWISE #.


#DF sequence-of-input-statement-units-in (stmt)

    "{ stmt #IS <input-statement> }"

    => #ERROR #.


#DF sequence-of-output-statement-units-in (stmt)

    "{ stmt #IS <output-statement> }"

    => #ERROR #.


#DF sequence-of-open-input-statement-units-in (stmt)

    "{ stmt #IS <open-input-statement> }"
```

```
     => #ERROR #.

#DF sequence-of-open-output-statement-units-in (stmt)

     "{ stmt #IS <open-output-statement> }"

     => #ERROR #.

#DF sequence-of-shut-input-statement-units-in (stmt)

     "{ stmt #IS <shut-input-statement> }"

     => #ERROR #.

#DF sequence-of-shut-output-statement-units-in (stmt)

     "{ stmt #IS shut-output-statement }"

     => #ERROR #.
```

```
#DF is-numeric-operation (nx)

     => #TRUE #IFF ($nx$) is-numeric-binary-operation #OR
        ($nx$) is-numeric-unary-operation #.


#DF is-operation-or-primitive-operand(nx)

     => #TRUE #IF ($nx$) is-a-constant;

     => #TRUE #IF ($nx$) is-a-variable;

     => #TRUE #IF ($nx$) is-numeric-binary-operation;

     => #TRUE #IF ($nx$) is-numeric-unary-operation;

     => #TRUE #IF ($nx$) is-boolean-operation;

     => #TRUE #IF ($nx$) is-relational-operation;

     => #TRUE #IF ($nx$) is-function-reference;

     => #FALSE #OTHERWISE #.


#DF is-a-constant (nx)

     => #TRUE #IFF nx #IS <floating-constant> #U
        <fixed-constant> #U <integer-constant> #U
        <boolean-constant> #U <octal-constant> #U
        <transmission-code-constant> #U <hollerith-constant>
        #U <status-constant> #U <zero> #.


#DF is-a-variable (nx)

     => #TRUE #IFF nx #IS <simple-variable> #U
        <indexed-variable> #U <entry-variable> #U
        <special-integer-variable> #U
        <special-fixed-variable> #U
        <special-literal-variable> #U
        <special-boolean-variable> #.


#DF is-numeric-binary-operation (nx)
```

```
     => #TRUE #IFF nx #IS <sum> #U<difference> #U<product>
        #U<quotient> #U<exponential> #.


#DF is-numeric-unary-operation (nx)

     => #TRUE #IFF nx #IS <abs-function> #U
        <nwdsen-function> #U <loc-function> #U <unary-minus>
        #U <unary-plus> #.


#DF is-boolean-operation (nx)

     => #TRUE #IFF nx #IS <disjunction> #U <conjunction> #U
        <negation> #.


#DF is-relational-operation (nx)

     => #TRUE #IFF nx #IS <entry-relation> #U
        <status-relation> #U <chain-relation> #U <relation>
        #.


#DF is-function-reference (unit)

     "{ ($unit$) is-operation-or-primitive-operand}"

     => #TRUE #IFF unit #IS <function-call> #.
```

```
#DF evaluation-successor-of (unit)

    "{unit #EQ current-executable-unit #AND ($unit$)
    is-operation-or-primitive-operand}"

    => successor-of-special-boolean-operand(unit) #IF
       ($unit$) is-special-boolean-operand;

    => successor-of-function-call (unit) #IF unit #IS
       <function-call>;

    => simple-successor-unit-of (unit) #OTHERWISE #.


#PROC-DF successor-of-function-call (unit)

    "{unit #IS <function-call>}"

    #COMPUTE! #ASSIGN-LATEST-VALUE
    (procedure-return-point-unique-to
    (procedure-decl-invoked-by(unit)), "receives" unit)

    #COMPUTE! assignment-to-input-parameters
    (procedure-decl-invoked-by (unit) , "arguments-from"
    unit)

    #RETURN-WITH-VALUE! #FIRST-ELEMENT-IN
    sequence-of-executable-procedure-units-in
    (procedure-decl-invoked-by (unit)) #.


#DF procedure-decl-invoked-by (unit)

    "{unit #IS <function-call>}"

    => procedure-dec-invoked-corresponding-to
       (declaration-for (function-name-in-reference
       (unit))) #.


#DF procedure-dec-invoked-corresponding-to (dec)

    "{dec #IS <procedure-declaration> #U
    <subprogram-declaration>>}"

    => dec #IF dec #IS <procedure-declaration>;
```

```
     => library-procedure-subprogram-corresponding-to (dec)
        #IF dec #IS <subprogram-declaration> #.


#DF library-procedure-subprogram-corresponding-to (dec)

     "{dec #IS <subprogram-declaration>}"

     => #FIRST proc #IN sequence-of-library-procedures-in
        (system-containing (dec)) #SUCH-THAT
        (name-declared-by (dec) #EQW name-of-proc (proc)) #.


#DF name-of-proc (dec)

     "{dec #IS <procedure-subprogram>}"

     => #SEG 3 #OF (procedure-head-of (dec)) #.


#DF sequence-of-library-procedures-in (sys)

     "{sys #IS <jovial-j3-system>}"

     => #SEQUENCE-OF <procedure-subprogram> #IN
        optional-library-of (sys) #.


#DF successor-of-special-boolean-operand(unit)

     "{ unit #EQ current-executable-unit #AND ($unit$)
     is-operation-or-primitive-operand #AND ($unit$)
     is-special-boolean-operand}"

     => innermost-special-boolean-operation-containing
        (unit) #IF
        innermost-special-boolean-operation-containing
        (unit) #IS <disjunction> #AND latest-value (unit)
        #EQ '1' "(true)";

     => innermost-special-boolean-operation-containing
        (unit) #IF
        innermost-special-boolean-operation-containing
        (unit) #IS <conjunction> #U<chain-relation> #AND
        latest-value (unit) #EQ '0' "(false)";

     => simple-successor-unit-of (unit) #OTHERWISE #.
```

```
#DF evaluation-effect-of (unit)

    "{unit #EQ current-executable-unit #AND
    ($unit$)is-operation-or-primitive-operand}"

    => evaluate-special-boolean-operand(unit) #IF ($unit$)
       is-special-boolean-operand;

    => evaluate (unit) #OTHERWISE #.


#DF evaluate (unit)

    "{unit #EQ current-executable-unit #AND ($unit$)
    is-operation-or-primitive-operand}"

    => #NIL #IF ($unit$)is-a-variable-not-to-be-evaluated
       #OR ($unit$)is-function-reference;

    => #NIL #IF ($unit$) is-special-boolean-operation;

    => assign-latest-value (unit, value(unit) ) #OTHERWISE
       #.


#DF latest-value (operand)

    "{($operand$)is-operation-or-primitive-operand}"

    => #LATEST-VALUE (unique-variable-corresponding-to
       (operand)) #.


#DF assign-latest-value (nx,val)

    "{nx #IS #NODE}"

    => #ASSIGN-LATEST-VALUE
       (unique-variable-corresponding-to (nx), "receives"
       val) #.


#PROC-DF evaluate-special-boolean-operand(unit)

    "{ unit #EQ current-executable-unit #AND ($unit$)
    is-operation-or-primitive-operand #AND ($unit$)
```

```
     is-special-boolean-operand}"

     #COMPUTE! evaluate(unit)

     #COMPUTE! #ASSIGN-LATEST-VALUE
     (unique-variable-corresponding-to
     (innermost-special-boolean-operation-containing
     (unit)), "receives" latest-value (unit))

     #RETURN-WITH-VALUE! #NIL #.


#DF is-a-variable-not-to-be-evaluated(unit)

     "{($unit$)is-operation-or-primitive-operand}"

     => ($unit$)is-not-to-be-evaluated #IF
        ($unit$)is-a-variable;

     => #FALSE #OTHERWISE #.


#DF is-not-to-be-evaluated (unit)

     "{ ($unit$) is-a-variable }"

     => #TRUE #IF ($unit$) is-a-receiving-variable-only;

     => #TRUE #IF ($unit$) is-call-by-name-argument;

     => #TRUE #IF ($variable-containing (unit) $)
        is-actual-output-data-parameter;

     => ($special-modifier-containing (unit) $)
        is-not-to-be-evaluated #IF ($unit$)
        is-modified-by-special-modifier;

     => #FALSE #OTHERWISE #.


#DF variable-containing (unit)

     => unit #IF #NOT #THERE-EXISTS nx #IN (
        #SEQUENCE-OF-ANCESTORS-OF unit) #SUCH-THAT (nx #IS
        <variable>) ;

     => #LAST nx #IN ( #SEQUENCE-OF-ANCESTORS-OF unit )
        #SUCH-THAT (nx #IS <variable>) #OTHERWISE #.
```

```
#DF is-modified-by-special-modifier (var)

    "{ ($var$) is-a-variable }"

    => var #EQ object-variable-of
       (special-modifier-containing (var)) #IF ($var$)
       is-contained-in-a-special-modifier;

    => #FALSE #OTHERWISE #.

#DF is-contained-in-a-special-modifier (var)

    "{ ($var$) is-a-variable }"

    => #THERE-EXISTS ancestor #IN (
       #SEQUENCE-OF-ANCESTORS-OF var) #SUCH-THAT
       (($ancestor$) is-a-special-modifier) #.

#DF is-a-special-modifier (nx)

    => #TRUE #IFF ($nx$) is-bit-functional-modifier #OR
       ($nx$) is-byte-functional-modifier #OR ($nx$)
       is-char-functional-modifier #OR ($nx$)
       is-mant-functional-modifier #OR ($nx$)
       is-odd-functional-modifier #.

#DF is-byte-functional-modifier (nx)

    => #TRUE #IFF nx #IS <special-literal-variable> #.

#DF is-mant-functional-modifier (nx)

    => #TRUE #IFF nx #IS <special-fixed-variable> #.

#DF is-odd-functional-modifier (nx)

    => #TRUE #IFF nx #IS <special-boolean-variable> #.

#DF special-modifier-containing (nx)
```

```
     => #LAST ancestor #IN ( #SEQUENCE-OF-ANCESTORS-OF nx)
        #SUCH-THAT ( ($ancestor$) is-a-special-modifier) #.


#DF is-a-receiving-variable-only(var)

     "{($var$)is-a-variable}"

     => var #EQ receiving-variable-of (left-hand-side-of
        (innermost-executable-statement-containing(var)))
        #IF innermost-executable-statement-containing(var)
        #IS <assignment-statement>;

     => #FALSE #OTHERWISE #.


#DF receiving-variable-of(nx)

     => #FIRST nx #IN (#SEQUENCE-OF-NODES-IN nx) #SUCH-THAT
        (($nx$)is-a-variable #OR nx #IS <name> ) #.


#DF is-special-boolean-operand(unit)

     "{unit #EQ current-executable-unit #AND ($unit$)
     is-operation-or-primitive-operand}"

     => #FALSE #IF #NOT ($unit$)
        is-contained-in-a-special-boolean-expression;

     => #TRUE #IF ($unit$) is-chain-relation-operand #OR
        ($unit$) is-conjunction-or-disjunction-operand;

     => #FALSE #OTHERWISE #.


#DF is-special-boolean-operation (unit)

     "{unit #EQ current-executable-unit #AND ($unit$)
     is-operation-or-primitive-operand}"

     => #TRUE #IFF unit #IS <conjunction> #U<disjunction>
        #U<chain-relation> #.


#DF innermost-special-boolean-operation-containing (unit)

     "{unit #EQ current-executable-unit #AND ($unit$)
```

```
    is-operation-or-primitive-operand #AND ($unit$)
    is-special-boolean-operand}"

    => #LAST-ELEMENT-IN
       sequence-of-special-boolean-operations-containing
       (unit) #.


#DF is-contained-in-a-special-boolean-expression (unit)

    "{unit #EQ current-executable-unit #AND ($unit$)
    is-operation-or-primitive-operand}"

    => #TRUE #IFF
       sequence-of-special-boolean-operations-containing
       (unit) #NEQ #NILSEQ #.


#DF is-chain-relation-operand(unit)

    "{unit #EQ current-executable-unit #AND ($unit$)
    is-operation-or-primitive-operand}"

    => #TRUE #IFF unit #IS <relation> #.


#DF is-conjunction-or-disjunction-operand(unit)

    "{unit #EQ current-executable-unit #AND ($unit$)
    is-operation-or-primitive-operand #AND ($unit$)
    is-contained-in-a-special-boolean-expression}"

    => #TRUE #IFF innermost-operation-containing (unit) #IS
       <conjunction> #U<disjunction> #.


#DF sequence-of-special-boolean-operations-containing (unit)

    "{($unit$) is-operation-or-primitive-operand}"

    => #SUBSEQUENCE-OF-ELEMENTS bool-op #IN
       sequence-of-operations-containing (unit) #SUCH-THAT
       (($bool-op$) is-special-boolean-operation) #.


#DF innermost-operation-containing (unit)

    "{($unit$) is-operation-or-primitive-operand #AND
```

```
     ($unit$) is-contained-in-a-special-boolean-operation}"

     => #LAST-ELEMENT-IN sequence-of-operations-containing
        (unit) #.

#DF sequence-of-operations-containing (unit)

     "{($unit$) is-operation-or-primitive-operand}"

     => #SUBSEQUENCE-OF-ELEMENTS op #IN
        sequence-of-evaluation-units-in
        (innermost-expression-containing (unit)) #SUCH-THAT
        (op #IS-IN #SEQUENCE-OF-ANCESTORS-OF (unit) ) #IF ($
        unit $) is-contained-in-an-expression ;

     => #NILSEQ #OTHERWISE #.

#DF innermost-expression-containing (unit)

     "{($unit$) is-operation-or-primitive-operand}"

     => #LAST fx #IN (#SEQUENCE-OF-ANCESTORS-OF (unit) )
        #SUCH-THAT (($fx$) is-an-evaluated-expression #OR ($
        fx $) is-an-evaluated-atomic-formula) #.

#DF is-contained-in-an-expression (unit)

     => #TRUE #IFF #THERE-EXISTS fx #IN
        #SEQUENCE-OF-ANCESTORS-OF (unit) #SUCH-THAT ( ($ fx
        $) is-an-evaluated-expression #OR ($ fx $)
        is-an-evaluated-atomic-formula ) #.

#DF is-an-evaluated-formula (nx)

     "{ nx #IS #NODE }"

     => nx #IS <boolean-formula> #U <numeric-formula> #U
        <formula> #.

#DF unique-variable-corresponding-to (unit)

     "{($unit$) is-operation-or-primitive-operand #AND unit
     #EQ current-executable-unit}"
```

```
    => 'evaluation-variable$' #CW ( #ORDPOSIT unit #IN
       (sequence-of-nodes-in ( #ROOT-NODE (unit)))) #.

#DF sequence-of-nodes-in (nx)

    "{ nx #IS #NODE }"

    => #SEQUENCE-OF-NODES-IN nx #.

#DF value (unit)

    "{($unit$) is-operation-or-primitive-operand #AND unit
    #EQ current-executable-unit}"

    => numeric-binary-operation-value (unit) #IF ($unit$)
       is-numeric-binary-operation;

    => numeric-unary-operation-value (unit) #IF ($unit$)
       is-numeric-unary-operation;

    => boolean-operation-value (unit) #IF ($unit$)
       is-boolean-operation;

    => relational-operation-value (unit) #IF ($unit$)
       is-relational-operation;

    => variable-value (unit) #IF ($unit$) is-a-variable;

    => constant-value (unit) #IF ($unit$) is-a-constant #.

#DF numeric-binary-operation-value (unit)

    "{($unit$) is-numeric-binary-operation #AND unit #EQ
    current-executable-unit}"

    => sum-value (unit) #IF unit #IS <sum>;

    => difference-value (unit) #IF unit #IS <difference>;

    => product-value (unit) #IF unit #IS <product>;

    => quotient-value (unit) #IF unit #IS <quotient>;

    => exponential-value (unit) #IF unit #IS <exponential>
```

```
    #.

#DF sum-value (unit)

    "{unit #IS <sum> #AND unit #EQ
    current-executable-unit}"

    => special-index-sum-value (unit) #IF ($unit$)
       is-special-index-sum;

    => integer-add (operand1-of(unit), "+"
       operand2-of(unit)) #IF type (unit) #EQW 'integer';

    => fixed-add (operand1-of(unit), "+" operand2-of(unit))
       #IF type (unit) #EQW 'fixed';

    => floating-add (operand1-of(unit), "+" operand2-of
       (unit)) #IF type (unit) #EQW 'floating' #.


#DF is-special-index-sum (op)

    "{op #EQ current-executable-unit}"

    => #FALSE #IF op #IS-NOT <sum>;

    => #FALSE #IF #NOT ($op$) is-contained-in-an-index;

    => #TRUE #IFF ($innermost-index-formula-containing
       (op)$) is-var-plus-or-minus-constant #OTHERWISE #.


#DF special-index-sum-value (unit)

    "{($unit$)is-special-index-sum}"

    => implementation-integer-add (integer-latest-value
       (operand1-of(unit)), "+" integer-latest-value
       (operand2-of(unit))) #.


#DF integer-add (operand1, "+" operand2)

    "{($operand1, operand2$)
    are-each-operation-or-primitive-operand}"

    => implementation-integer-add (latest-value (operand1),
```

```
        "+" latest-value (operand2)) #.

#DF fixed-add (operand1, "+" operand2)

     "{($operand1, operand2$)
     are-each-operation-or-primitive-operand}"

     => implementation-fixed-add (latest-value (operand1),
        "with" attributes (operand1), "+" latest-value
        (operand2), "with" attributes(operand2),
        "producing-a-result-with" attributes (sum-containing
        (operand1, "and" operand2))) #.

#DF floating-add (operand1, "+" operand2)

     "{($operand1, operand2$)
     are-each-operation-or-primative-operand}"

     => implementation-floating-add(
        latest-value(operand1),"+" latest-value(operand2))#.

#DF integer-latest-value(operand)

     "{($operand$) is-operation-or-primitive-operand}"

     => latest-value (operand) #IF type (operand) #EQW
        'integer';

     => ($ latest-value (operand), "with" attributes
        (operand) $) converted-fixed-to-integer #IF type
        (operand) #EQW 'fixed';

     => ($ latest-value (operand) $)
        converted-floating-to-integer #IF type(operand) #EQW
        'floating' #.

#DF is-contained-in-an-index (op)

     "{op #IS <sum> #U <difference>}"

     => #TRUE #IFF #THERE-EXISTS nx #IN (
        #SEQUENCE-OF-ANCESTORS-OF (op) ) #SUCH-THAT (nx #IS
        <index-formula>) #.
```

```
#DF innermost-index-formula-containing (op)

     "{op #IS <sum> #U <difference> #AND ($op$)
     is-contained-in-an-index}"

     => #LAST nx #IN ( #SEQUENCE-OF-ANCESTORS-OF (op) )
        #SUCH-THAT (nx #IS <index-formula>) #.


#DF is-var-plus-or-minus-constant (nx)

     "{nx #IS <index-formula>}"

     => #FALSE #IF formula-comprising (nx) #IS
        <atomic-numeric-formula>;

     => #FALSE #IF outermost-numeric-operation-in (nx)
        #IS-NOT <sum> #U <difference>;

     => #TRUE
        #IFF($operand1-of(outermost-numeric-operation-in
        (nx))$)is-a-variable #AND ($operand2-of
        (outermost-numeric-operation-in (nx ))$)
        is-a-constant #OTHERWISE #.


#DF sum-containing (operand1, operand2)

     "{($operand1, operand2$)
     are-each-operation-or-primitive-operand}"

     => #LAST nx #IN (#SEQUENCE-OF-ANCESTORS-OF (operand1) )
        #SUCH-THAT (nx #IS <sum>) #.


#DF outermost-numeric-operation-in (nx)

     "{nx #IS <index-formula>}"

     => #FIRST node #IN ( #SEQUENCE-OF-NODES-IN (nx) )
        #SUCH-THAT (($node$) is-numeric-operation) #.


#DF difference-value (unit)

     "{unit #IS <difference> #AND unit #EQ
     current-executable-unit}"
```

```
     => special-index-difference-value (unit) #IF ($unit$)
        is-special-index-difference;

     => integer-subtract (operand1-of(unit), "-" operand2-of
        (unit)) #IF type (unit) #EQW 'integer';

     => fixed-subtract (operand1-of(unit), "-" operand2-of
        (unit)) #IF type (unit) #EQW 'fixed';

     => floating-subtract (operand1-of(unit),"-"
        operand2-of(unit)) #IF type (unit) #EQW 'floating'
        #.


#DF is-special-index-difference(op)

     "{op #EQ current-executable-unit}"

     => #FALSE #IF op #IS-NOT <difference>;

     => #FALSE #IF #NOT ($op$) is-contained-in-an-index;

     => #TRUE #IFF ($innermost-index-formula-containing
        (op)$) is-var-plus-or-minus-constant #OTHERWISE #.


#DF special-index-difference-value (unit)

     "{($unit$) is-special-index-difference}"

     => implementation-integer-subtract
        (integer-latest-value (operand1-of(unit)), "-"
        integer-latest-value (operand2-of(unit))) #.


#DF integer-subtract (operand1, operand2)

     "{($operand1, operand2$)
     are-each-operation-or-primitive-operand}"

     => implementation-integer-subtract (latest-value
        (operand1), "-" latest-value (operand2)) #.


#DF fixed-subtract (operand1, "-" operand2)

     "{($operand1, operand2$)
```

```
    are-each-operation-or-primitive-operand}"

    => implementation-fixed-subtract (latest-value
       (operand1), "with" attributes (operand1), "-"
       latest-value (operand2), "with" attributes
       (operand2), "producing-a-result-with" attributes
       (difference-containing (operand1, "and" operand2)))
       #.


#DF floating-subtract(operand1, "-" operand2)

    "{($operand1,operand2$)
    are-each-operation-or-primative-operand}"

    => implementation-floating-subtract
       (latest-value(operand1), "-" latest-value(operand2))
       #.


#DF difference-containing (operand1, operand2)

    "{($operand1, operand2$)
    are-each-operation-or-primitive-operand}"

    => #LAST nx #IN (#SEQUENCE-OF-ANCESTORS-OF (operand1) )
       #SUCH-THAT (nx #IS <difference>) #.


#DF product-value (unit)

    "{unit #IS <product> #AND unit #EQ
    current-executable-unit}"

    => integer-product (operand1-of(unit), "*"
       operand2-of(unit)) #IF type (unit) #EQW 'integer';

    => fixed-product (operand1-of(unit), "*"
       operand2-of(unit)) #IF type (unit) #EQW 'fixed';

    => floating-product (operand1-of(unit), "*"
       operand2-of(unit) ) #IF type (unit) #EQW 'floating'
       #.


#DF integer-product (operand1, "*" operand2)

    "{($operand1, operand2$)
```

```
    are-each-operation-or-primitive-operand}"

    => implementation-integer-product (latest-value
       (operand1), "*" latest-value (operand2)) #.


#DF fixed-product (operand1, "*" operand2)

    "{($operand1, operand2$)
    are-each-operation-or-primitive-operand}"

    => implementation-fixed-product (latest-value
       (operand1), "with" attributes (operand1), "*"
       latest-value (operand2), "with" attributes
       (operand2), "producing-a-result-with" attributes
       (product-containing (operand1, "and" operand2))) #.


#DF floating-product(operand1, "*" operand2)

    "{($operand1,operand2$)
    are-each-operation-or-primative-operand}"

    => implementation-floating-product
       (latest-value(operand1), "*" latest-value(operand2))
       #.


#DF product-containing (operand1, operand2)

    "{($operand1, operand2$)
    are-each-operation-or-primitive-operand}"

    => #LAST nx #IN (#SEQUENCE-OF-ANCESTORS-OF (operand1) )
       #SUCH-THAT (nx #IS <product>) #.


#DF quotient-value (unit)

    "{unit #IS <quotient> #AND unit #EQ
    current-executable-unit}"

    => fixed-quotient (operand1-of(unit), "/"
       operand2-of(unit)) #IF type (unit) #EQW 'fixed';

    => floating-quotient (operand1-of(unit), "/"
       operand2-of(unit)) #IF type (unit) #EQW 'floating'
       #.
```

```
#DF fixed-quotient (operand1, "/" operand2)

    "{($operand1, operand2$)
    are-each-operation-or-primitive-operand}"

    => implementation-fixed-quotient (latest-value
       (operand1), "with" attributes (operand1), "/"
       latest-value (operand2), "with" attributes
       (operand2), "producing-a-result-with" attributes
       (quotient-containing (operand1, "and" operand2))) #.


#DF floating-quotient(operand1, "/" operand2)

    "{($operand1,operand2$)
    are-each-operation-or-primative-operand}"

    => implementation-floating-quotient
       (latest-value(operand1),"/" latest-value(operand2))
       #.


#DF quotient-containing (operand1, operand2)

    "{($operand1, operand2$)
    are-each-operation-or-primitive-operand}"

    => #LAST nx #IN (#SEQUENCE-OF-ANCESTORS-OF (operand1) )
       #SUCH-THAT (nx #IS <quotient>) #.


#DF exponential-value (unit)

    "{unit #IS <exponential> #AND unit #EQ
    current-executable-unit}"

    => special-exponential-value (unit) #IF ($unit$)
       is-special-exponential;

    => floating-exponential-value (unit) #OTHERWISE #.


#DF is-special-exponential (op)

    "{op #IS <exponential> #AND unit #EQ
    current-executable-unit}"
```

```
    => #FALSE #IF operand2-of(op) #IS-NOT
       <integer-constant>;

    => #FALSE #IF type (operand1-of(op)) #EQW 'floating';

    => #TRUE #IF #STRING-OF-TERMINALS-OF (operand2-of(op))
       * size-of-result-of (operand1-of(op)) <
       bits-per-word;

    => #FALSE #OTHERWISE #.


#DF special-exponential-value(unit)

    "{unit#IS <exponential> #AND unit #EQ
    current-executable-unit}"

    => special-exponential(operand1-of(unit), "**"
       operand2-of(unit)) #.


#DF special-exponential(operand1, "**"operand2)

    "{($operand1,operand2$)
    are-each-operation-or-primative-operand}"

    => implementation-special-exponential
       (latest-value(operand1),"**" latest-value(operand2))
       #.


#DF floating-exponential-value(unit)

    "{unit #IS <exponential> #AND unit #EQ
    current-executable-unit}"

    => floating-exponential(operand1-of(unit), "**"
       operand2-of(unit)) #.


#DF floating-exponential(operand1,"**"operand2)

    "{($operand1,operand2$)
    are-each-operation-or-primative-operand}"

    => implementation-floating-exponential
       (latest-value(operand1),"**" latest-value(operand2))
```

```
    #.

#DF numeric-unary-operation-value (unit)

     "{($unit$) is-numeric-unary-operation #AND unit #EQ
     current-executable-unit}"

     => abs-function-value (unit) #IF unit #IS
        <abs-function>;

     => nwdsen-function-value (unit) #IF unit #IS
        <nwdsen-function>;

     => loc-function-value (unit) #IF unit #IS
        <loc-function>;

     => unary-minus-value (unit) #IF unit #IS <unary-minus>;

     => unary-plus-value (unit) #IF unit #IS <unary-plus> #.


#DF abs-function-value (unit)

     "{unit #IS <abs-function> #AND unit #EQ
     current-executable-unit}"

     => ($standard-magnitude (latest-value (operand1-of
        (unit))) #CW '#B2'$)
        with-result-converted-to-implementation-form #.


"The largest negative number becomes zero."


#DF nwdsen-function-value(unit)

     "{unit #IS <nwdsen-function>}"

     => ($ #CONVERT 2 (number-of-words-per-entry-in
        (declaration-for (tabular-name-of
        (unit))))$)with-result-converted-to-implementation-form
        #.


#DF loc-function-value (unit)

     => ($ #CONVERT 2( word-address-from
```

```
        (standard-reference-address-of
        (loc-name-of(unit))))$)converted-to-implementation-form
        #.


#DF loc-name-of (unit)

     "{unit #IS <loc-function>}"

     => #SEG 5 #OF unit #IF unit #IS #CASE 1 #OF
        <loc-function> #.


#DF unary-minus-value (unit)

     "{unit #IS <unary-minus> #AND unit #EQ
     current-executable-unit}"

     => ($ #NEG ($latest-value (operand1-of (unit))$)
        converted-to-standard-form$)
        with-result-converted-to-implementation-form #.


#DF unary-plus-value (unit)

     "{unit #IS <unary-plus> #AND unit #EQ
     current-executable-unit}"

     => unit #.


#DF boolean-operation-value (unit)

     "{($unit$) is-boolean-operation #AND unit #EQ
     current-executable-unit}"

     => disjunction-value (unit) #IF unit #IS <disjunction>;

     => conjunction-value (unit) #IF unit #IS <conjunction>;

     => negation-value (unit) #IF unit #IS <negation> #.


#DF disjunction-value (unit)

     "{unit #IS <disjunction> #AND unit #EQ
     current-executable-unit}"
```

```
    => latest-value (unit) #.

#DF conjunction-value (unit)

    "{unit #IS <conjunction> #AND unit #EQ
    current-executable-unit}"

    => latest-value (unit) #.

#DF negation-value (unit)

    "{unit #IS <negation> #AND unit #IS
    current-executable-unit}"

    => '1' #IF latest-value (operand1-of (unit)) #EQW '0';

    => '0' #OTHERWISE #.

#DF relational-operation-value (unit)

    "{($unit$) is-relational-operation #AND unit #EQ
    current-executable-unit}"

    => entry-relation-value (unit) #IF unit #IS
       <entry-relation>;

    => status-relation-value (unit) #IF unit #IS
       <status-relation>;

    => chain-relation-value (unit) #IF unit #IS
       <chain-relation>;

    => relation-value (unit) #IF unit #IS <relation> #.

#DF entry-relation-value (unit)

    "{unit #IS <entry-relation>}"

    => '1' #IF ($ entry-compare-value(unit), "and"
       #STRING-OF-TERMINALS-OF
       (relation-constant-of(unit))$) are-in-agreement;

    => '0' #OTHERWISE #.
```

```
#DF entry-compare-value(unit)

     "{unit #IS <entry-relation>}"

     => entry-compare (latest-value (operand1-of(unit)),"to"
        latest-value (operand2-of(unit)))#.

#DF entry-compare(val1,val2)

     "{ ($\val1,val2\$)are-strings-of-ones-and-zeroes}"

     => ($#SIGN(standard-rep(val1) - standard-rep(val2))$)
        converted-to-rel-symbol #.

#DF chain-relation-value (unit)

     "{unit #IS <chain-relation>}"

     => latest-value (unit) #.

#DF relation-value (unit)

     "{unit #IS <relation> #AND unit #EQ
     current-executable-unit}"

     => status-relation-value (unit) #IF ($unit$)
        is-actually-a-status-relation;

     => literal-relation-value (unit) #IF ($unit$)
        is-actually-a-literal-relation;

     => numeric-relation-value (unit) #IF ($unit$)
        is-actually-a-numeric-relation #.

#DF is-actually-a-status-relation (unit)

     "{unit #IS <relation>}"

     => #TRUE #IFF ($operand1-of(unit)$)
        is-file-name-or-status-variable #AND
        ($operand2-of(unit)$) is-a-status-formula #.
```

```
#DF is-file-name-or-status-variable (nx)

    "{($nx$)is-operation-or-primitive-operand}"

    => #FALSE #IF nx #IS-NOT <simple-variable> #U
       <indexed-variable>;

    => #TRUE #IFF ($detailed-declaration-for (nx)$)
       is-file-or-status-declaration #OTHERWISE#.


#DF is-file-or-status-declaration(decl)

    "{decl #IS <simple-item-declaration> #U
    <array-declaration> #U <ordinary-table-declaration> #U
    <defined-entry-table-declaration> #U
    <like-table-declaration> #U
    <ordinary-table-item-declaration> #U
    <string-item-declaration> #U
    <defined-entry-item-declaration> #U <file-declaration>
    #U <procedure-declaration> #U <subprogram-declaration>
    #U <mode-directive>}"

    => #TRUE #IF decl #IS <file-declaration>;

    => ($decl$)is-status-declaration
       #IF($decl$)is-typed-data-declaration;

    => #FALSE #OTHERWISE #.


#DF is-typed-data-declaration (nx)

    "{#IS #NODE}"

    => #TRUE #IFF nx #IS <simple-item-declaration> #U
       <array-declaration> #U
       <ordinary-table-item-declaration> #U
       <string-item-declaration> #U
       <defined-entry-item-declaration> #U <mode-directive>
       #.


#DF is-status-declaration (decl)

    "{($decl$)is-typed-data-declaration #OR decl #IS
    <procedure declation>}"
```

```
     => #TRUE #IFF declaration-type (decl) #EQW 'status' #.


#DF is-a-status-formula(nx)

     "{($nx$) is-operation-or-primitive-operand}"

     => #TRUE #IF nx #IS <status-constant> ;

     => ($detailed-declaration-for (nx)$)
        is-status-declaration #IF
        ($nx$)is-typed-data-reference;

     => function-type(nx) #EQW 'status' #IF($nx$)
        is-function-reference #.


#DF is-typed-data-reference (nx)

     "{ ($nx$) is-operation-or-primitive-operand}"

     => #FALSE #IF nx #IS-NOT <simple-variable> #U
        <indexed-variable>;

     => #TRUE#IFF ($ detailed-declaration-for (nx)$)
        is-typed-data-declaration #OTHERWISE#.


#DF status-relation-value (srel)

     "{srel #IS <status-relation> #OR srel #IS <relation>
     #AND ($srel$) is-actually-a-status-relation}"

     => '1' #IF ($status-compare-value (srel), "and"
        #STRING-OF-TERMINALS-OF (relation-constant-of
        (srel))$) are-in-agreement;

     => '0' #OTHERWISE #.


#DF status-compare-value (srel)

     => implementation-status-compare (latest-value
        (operand1-of(srel)),"with + "
        latest-value(operand2-of(srel)))#.


#DF implementation-status-compare (valx,valy)
```

```
    => relation-with-integer-zero-of
       (implementation-integer-subtract (valx, "-" valy))
       #.

#DF is-actually-a-literal-relation (unit)

    "{ unit #IS <relation>}"

    => #TRUE #IF ($operand2-of(unit)$)is-literal-object;

    => #TRUE #IF ($operand1-of(unit)$)is-literal-object;

    => #TRUE #IFF
       ($operand1-of(unit)$)is-actually-a-literal-relation
       #OTHERWISE#.

#DF is-literal-object(unit)

    "{($unit$)is-operation-or-primitive-operand}"

    => #TRUE #IFF type (unit) #IS-IN
       \'hollerith','transmission-code'\ #.

#DF literal-relation-value (unit)

    "{unit #IS <relation> #AND ($unit$)
    is-actually-a-literal-relation}"

    => '1' #IF ($literal-compare-value (unit),"and"
       #STRING-OF-TERMINALS-OF
       (relation-constant-of(unit))$) are-in-agreement;

    => '0' #OTHERWISE#.

#DF literal-compare-value (unit)

    "{ unit #IS <relation> #AND
    ($unit$)is-actually-a-literal-relation}"

    => literal-compare (normalized-literal (left-operand-of
       (unit), "wrt" right-operand-of (unit)), "to"
       normalized-literal (right-operand-of (unit), "wrt"
       left-operand-of (unit))) #.
```

```
#DF normalized-literal (operand1,operand2)

     "{type(operand1)#IS-IN
     \'hollerith','transmission-code','octal'\#AND type
     (operand2) #IS-IN
     \'hollerith','transmission-code','octal'\}"

     => latest-value (operand1) #IF #LENGTH (
        latest-value(operand1)) >= #LENGTH(
        latest-value(operand2));

     => padded-literal-value (operand1,"to" #LENGTH (
        latest-value (operand2)) - #LENGTH
        (latest-value(operand1)) "bits") #OTHERWISE#.

#DF padded-literal-value (operand, "to" n "bits")

     "{ n>0 #AND type(operand)
     #IS-IN\'hollerith','transmission-code', 'octal'\}"

     => ($n$)bits-worth-of-hollerith-blanks #CW
        latest-value(operand) #IF type (operand) #EQW
        'hollerith';

     => ($n$) bits-worth-of-transmission-code-blanks #CW
        latest-value(operand) #IF type (operand) #EQW
        'transmission-code';

     => ($n$) zeroes #CW latest-value(operand) #IF type
        (operand) #EQW 'octal' #.

#DF bits-worth-of-hollerith-blanks (n)

     "{n>0}"

     => replicate ( hollerith-for(#SPACE), (n - 1) /
        bits-per-byte + 1 "times" ) #.

#DF bits-worth-of-transmission-code-blanks (n)

     "{n>0}"

     => replicate (transmission-code-for (#SPACE), (n - 1) /
```

```
      bits-per-byte + 1 "times") #.

#DF replicate (str, n "times")

     "{n>=0}"

     => #NIL #IF n=0;

     => str #CW replicate (str, n - 1 "times") #OTHERWISE #.

#DF literal-compare(val1,val2)

     "{($\val1,val2\$)are-strings-of-ones-and-zeroes}"

     => ($ #SIGN (standard-rep(val1) - standard-rep(val2))$)
        converted-to-rel-symbol #.

#DF converted-to-rel-symbol (i)

     "{i #IS-IN \-1,0,1\}"

     => '=' #IF i=0;

     => '<' #IF i=-1;

     => '>' #IF i=1 #.

#DF standard-rep(val)

     "{($val$) is-string-of-ones-and-zeroes}"

     => val #CW '#B2' #.

#DF is-actually-a-numeric-relation (unit)

     "{ unit #IS <relation> #AND #NOT ($unit$)
     is-actually-a-status-relation #AND #NOT ($unit$)
     is-actually-a-literal-relation}"

     => #TRUE #IF ($operand2-of(unit)$)is-numeric-object;

     => #TRUE #IF ($operand1-of(unit)$)is-numeric-object;
```

```
     => #TRUE #IFF
        ($operand1-of(unit)$)is-actually-a-numeric-relation
        #OTHERWISE #.


#DF is-numeric-object (unit)

     "{ ($unit$) is-operation-or-primitive-operand}"

     => #TRUE #IFF type(unit) #IS-IN \ 'integer','fixed',
        'floating'\ #.


#DF numeric-relation-value (unit)

     "{ ($ unit $) is-actually-a-numeric-relation #AND unit
     #EQ current-executable-unit }"

     => '1' #IF ($ numeric-compare-value (unit), "and"
        #STRING-OF-TERMINALS-OF (relation-constant-of (unit)
        ) $) are-in-agreement;

     => '0' #OTHERWISE #.


#DF are-in-agreement (compare-result, rel-const)

     "{ compare-result #IS-IN \'<','>','='\ #AND rel-const
     #IS-IN \'EQ','GR','GQ','LQ','LS','NQ'\ }"

     => #TRUE #IFF compare-result #EQW '<' #AND rel-const
        #IS-IN \'LQ','LS','NQ'\ #OR compare-result #EQW '>'
        #AND rel-const #IS-IN \'GR','GQ','NQ'\ #OR
        compare-result #EQW '=' #AND rel-const #IS-IN
        \'EQ','GQ','LQ'\ #.


#DF numeric-compare-value (unit)

     "{unit #IS <relation> #AND unit #EQ
     current-executable-unit}"

     => implementation-floating-compare(
        floating-latest-value (left-operand-of (unit)),
        "with" floating-latest-value
        (right-operand-of(unit)) ) #IF
        either-operand-is-float-in (unit) #EQW 'floating';
```

```
    => implementation-integer-and-fixed-point-compare
       (latest-value (left-operand-of(unit)), "with"
       attributes (left-operand-of(unit)), latest-value
       (right-operand-of(unit)), "with" attributes
       (right-operand-of(unit))) #OTHERWISE #.


#DF left-operand-of (rel)

    "{rel #IS <relation>}"

    => operand1-of (rel) #IF operand1-of(rel) #IS-NOT
       <relation>;

    => operand2-of (operand1-of (rel)) #OTHERWISE #.


#DF right-operand-of(rel)

    "{rel #IS <relation>}"

    => operand2-of (rel) #.


#DF variable-value (unit)

    "{($unit$) is-a-variable #AND unit #EQ
    current-executable-unit}"

    => generalized-latest-value (
       standard-reference-address-of(unit)) #.


#DF constant-value (unit)

    "{ ($unit$) is-a-constant #AND unit #EQ
    current-executable-unit }"

    => floating-constant-value (unit) #IF unit #IS
       <floating-constant>;

    => fixed-constant-value (unit) #IF unit #IS
       <fixed-constant>;

    => integer-constant-value (unit) #IF unit #IS
       <integer-constant>;

    => boolean-constant-value (unit) #IF unit #IS
```

```
        <boolean-constant>;

    => octal-constant-value (unit) #IF unit #IS
       <octal-constant>;

    => transmission-code-constant-value (unit) #IF unit #IS
       <transmission-code-constant>;

    => hollerith-constant-value (unit) #IF unit #IS
       <hollerith-constant>;

    => status-constant-value (unit) #IF unit #IS
       <status-constant>;

    => zero-constant-value (unit) #IF unit #IS <zero> #.


#DF floating-constant-value (unit)

    "{unit #IS <floating-constant> #AND unit #EQ
    current-executable-unit}"

    => implementation-floating-zero #IF
       integer-portion-indicated-in-floating-constant
       (unit)=0 #AND
       fraction-portion-indicated-in-floating-constant
       (unit)=0;

    => ($ \ binary-exponent-for (unit),
       double-word-representation-of
       (binary-mantissa-for(unit))\$)
       converted-to-implementation-floating-form #OTHERWISE
       #.


#DF fixed-constant-value (unit)

    "{unit #IS <fixed-constant>}"

    => ($ integer-portion-value (unit) #CW
       fractional-portion-value
       (unit)$)conformed-to-implementation-word-size #IF
       fractional-bits-in-result-of (unit) >= 0;

    => implementation-right-arithmetic-shift
       (integer-portion-value (unit) ,"by" #ABS
       (fractional-bits-in-result-of (unit))) #IF
       fractional-bits-in-result-of (unit) <0 #.
```

```
#DF integer-portion-value (unit)

     "{unit #IS <fixed-constant> #U <integer-constant> #AND
     unit #EQ current-executable-unit}"

     => ($ #CONVERT 2 (integer-portion-indicated-in
        (unit))$)
        with-result-converted-to-implementation-form #.

#DF integer-portion-indicated-in (unit)

     "{unit #IS <fixed-constant> #U <integer-constant>}"

     => decimal-representation-of (unit) #IF unit #IS
        <integer-constant>;

     => integer-portion-indicated-in-fixed-constant (unit)
        #IF unit #IS <fixed-constant> #.

#DF decimal-representation-of (unit)

     "{unit #IS <integer-constant>}"

     => ( #STRING-OF-TERMINALS-OF (integer-part-of (unit)) )
        #CW zeroes-indicated-by-exponent-of (unit) #.

#DF zeroes-indicated-by-exponent-of (unit)

     "{unit #IS <integer-constant>}"

     => ($ #STRING-OF-TERMINALS-OF (exponent-part-of (unit)
        ) $) zeroes #IF ($unit$) has-an-exponent-part;

     => #NIL #OTHERWISE #.

#DF has-an-exponent-part (nx)

     "{nx #IS #NODE}"

     => #TRUE #IFF nx #IS <exponentiated-floating-constant>
        #OR nx #IS #CASE 2 #OF <integer-constant> #.
```

```
#DF zeroes (num)

    "{0 <= num}"

    => #LEFT num #CHARACTERS-OF string-of-zeroes #IF num <=
       #LENGTH (string-of-zeroes) ;

    => string-of-zeroes #CW ($num - #LENGTH
       (string-of-zeroes) $) zeroes #OTHERWISE #.


#DF string-of-zeroes

    => '000000000000000000000000000000000000' #.


#DF ones(num)

    "{ 0 <= num }"

    => #LEFT num #CHARACTERS-OF string-of-ones #IF num <=
       #LENGTH (string-of-ones) ;

    => string-of-ones #CW ($ num - #LENGTH (string-of-ones)
       $) ones #OTHERWISE #.


#DF string-of-ones

    => '111111111111111111111111111111111111' #.


#DF integer-portion-indicated-in-fixed-constant (unit)

    "{unit #IS <fixed-constant>}"

    => simple-integer-portion-from (floating-part-of
       (unit)) #IF floating-part-of (unit) #IS
       <simple-floating-constant>;

    => exponentiated-integer-portion-from (floating-part-of
       (unit)) #IF floating-part-of (unit) #IS
       <exponentiated-floating-constant> #.


#DF simple-integer-portion-from (nx)
```

```
     "{nx #IS <simple-floating-constant> #U
     <exponentiated-floating-constant>}"

     => #STRING-OF-TERMINALS-OF (integer-part-of (nx)) #IF
        ($nx$) has-an-integer-part;

     => '0' #OTHERWISE #.


#DF exponentiated-integer-portion-from (nx)

     "{nx #IS <exponentiated-floating-constant>}"

     => ($simple-integer-portion-from (nx), "by" #ABS
        (exponent-part-of (nx))$)
        decimal-integer-right-shifted #IF exponent-part-of
        (nx) <=0;

     => simple-integer-portion-from (nx) #CW
        digits-shifted-in-from-fractional-part-of (nx)
        #OTHERWISE #.


#DF digits-shifted-in-from-fractional-part-of (nx)

     "{nx #IS <exponentiated-floating-constant> #AND
     exponent-part-of (nx) > 0}"

     => #LEFT exponent-part-of (nx) #CHARACTERS-OF
        (simple-fractional-portion-from (nx) #CW
        ($exponent-part-of (nx)$) zeroes) #.


#DF simple-fractional-portion-from (nx)

     "{nx #IS <simple-floating-constant > #U
     <exponentiated-floating-constant>}"

     => #STRING-OF-TERMINALS-OF (fraction-part-of (nx)) #IF
        ($nx$) has-a-fraction-part;

     => '0' #OTHERWISE #.


#DF decimal-integer-right-shifted (num, "by" count)

     "{num #IS-IN #NAT-NOS #AND count #IS-IN #NAT-NOS}"
```

```
        => '0' #IF count >= #LENGTH (num);

        => #LEFT ( #LENGTH (num) - count) #CHARACTERS-OF num
        #OTHERWISE #.

#DF fraction-part-of (nx)

        "{nx #IS <simple-floating-constant> #U
        <exponential-floating-constant> #AND ($nx$)
        has-a-fraction-part}"

        => fraction-part-of (simple-floating-constant-part-of
           (nx)) #IF nx #IS <exponentiated-floating-constant>;

        => #SEG 3 #OF nx #IF nx #IS #CASE 2 #OF
           <simple-floating-constant> ;

        => #SEG 2 #OF nx #IF nx #IS #CASE 3 #OF
           <simple-floating-constant> #.

#DF fractional-portion-value (unit)

        "{unit #IS <fixed-constant> #AND unit #EQ
        current-executable-unit}"

        => fractional-binary-representation-of
           (fraction-portion-indicated-in-fixed-constant
           (unit), "to" fractional-bits-in-result-of (unit)
           "binary-digits") #.

#DF fraction-portion-indicated-in-fixed-constant (nx)

        "{nx #IS <fixed-constant>}"

        => simple-fractional-portion-from (floating-part-of
           (nx)) #IF floating-part-of (nx) #IS
           <simple-floating-constant>;

        => exponentiated-fractional-portion-from
           (floating-part-of (nx)) #IF floating-part-of (nx)
           #IS <exponentiated-floating-constant> #.

#DF exponentiated-fractional-portion-from (nx)
```

```
    "{nx #IS <exponentiated-floating-constant>}"

    => ($ simple-fractional-portion-from (nx), "by"
       exponent-part-of (nx)$)decimal-fraction-left-shifted
       #IF exponent-part-of (nx) >=0;

    => digits-shifted-in-from-integer-part-of (nx) #CW
       simple-fractional-portion-from (nx) #OTHERWISE #.


#DF decimal-fraction-left-shifted (num, "by" count)

    "{num #IS-IN #NAT-NOS #AND count #IS-IN #NAT-NOS}"

    => '0' #IF count >= #LENGTH (num);

    => #RIGHT ( #LENGTH (num) - count) #CHARACTERS-OF num
       #OTHERWISE #.


#DF digits-shifted-in-from-integer-part-of (nx)

    "{nx #IS <exponentiated-floating-constant> #AND
    exponent-part-of (nx) < 0}"

    => #RIGHT #ABS (exponent-part-of (nx)) #CHARACTERS-OF
       (($ #ABS(exponent-part-of (nx))$) zeroes #CW
       simple-integer-portion-from(nx)) #.


#DF integer-constant-value (unit)

    "{unit #IS <integer-constant>}"

    => integer-portion-value (unit) #.


#DF boolean-constant-value (unit)

    "{unit #IS <boolean-constant>}"

    => #STRING-OF-TERMINALS-OF (unit) #.


#DF octal-constant-value (unit)

    "{unit #IS <octal-constant>}"
```

```
    => ($ #CONVERT 10 (digits-of (unit)) $)
       converted-to-implementation-form #.


#DF transmission-code-constant-value (unit)

    "{unit #IS <transmission-code-constant> #AND unit #EQ
    current-executable-unit}"

    => transmission-code-value-for (
       #STRING-OF-TERMINALS-OF (transmission-code-text-of
       (unit))) #.


#DF transmission-code-text-of (unit)

    "{ unit #IS <transmission-code-constant> #AND unit #EQ
    current-executable-unit }"

    => #SEG 4 #OF unit #.


#DF transmission-code-value-for (str)

    "{str #IS <sign-string>}"

    => #NIL #IF #LENGTH (str) = 0;

    => transmission-code-for ( #FIRST-CHARACTER-IN (str) )
       #CW transmission-code-value-for
       (all-but-first-character-in (str)) #OTHERWISE #.


#DF all-but-first-character-in (str)

    "{str #IS #STRING}"

    => #RIGHT ( #LENGTH (str) - 1) #CHARACTERS-OF str #.


#DF transmission-code-for (byte)

    "{byte #IS <sign>}"

    => ($
       six-bit-binary-representation-of-transmission-code-for
       (byte)$) with-filler-bits-added #.
```

```
#DF six-bit-binary-representation-of-transmission-code-for
(byte)

     "{byte #IS <sign>}"

     => #PREFIX-OF-FIRST '#B2'#IN ( #CONVERT 2 (2
        #TH-ELEMENT-IN ( #FIRST map-pair #IN
        transmission-code-map-pair-sequence #SUCH-THAT (
        #FIRST-ELEMENT-IN map-pair #EQW byte)))) #.

#DF with-filler-bits-added (code)

     "{($code$) is-string-of-ones-and-zeroes}"

     => ($bits-per-byte - #LENGTH (code)$) zeroes #CW code
        #.

#DF transmission-code-map-pair-sequence

     => \
          \#SPACE, '00#B8' \, \ 'A', '06#B8' \, \ 'B',
        '07#B8' \,
          \ 'C', '10#B8' \, \ 'D', '11#B8' \, \ 'E',
        '12#B8' \,
          \ 'F', '13#B8' \, \ 'G', '14#B8' \, \ 'H',
        '15#B8' \,
          \ 'I', '16#B8' \, \ 'J', '17#B8' \, \ 'K',
        '20#B8' \,
          \ 'L', '21#B8' \, \ 'M', '22#B8' \, \ 'N',
        '26#B8' \,
          \ 'O', '24#B8' \, \ 'P', '25#B8' \, \ 'Q',
        '26#B8' \,
          \ 'R', '27#B8' \, \ 'S', '30#B8' \, \ 'T',
        '31#B8' \,
          \ 'U', '32#B8' \, \ 'V', '33#B8' \, \ 'W',
        '34#B8' \,
          \ 'X', '35#B8' \, \ 'Y', '36#B8' \, \ 'Z',
        '37#B8' \,
          \ ')', '40#B8' \, \ '-', '41#B8' \, \ '+',
        '42#B8' \,
          \ '=', '44#B8' \, \ '$', '47#B8' \, \ '*',
        '50#B8' \,
          \ '(', '51#B8' \, \ ',', '56#B8' \, \ '0',
        '60#B8' \,
          \ '1', '61#B8' \, \ '2', '62#B8' \, \ '3',
```

```
    '63#B8' \,
       \ '4', '64#B8' \, \ '5', '65#B8' \, \ '6',
    '66#B8' \,
       \ '7', '67#B8' \, \ '8', '70#B8' \, \ '9',
    '71#B8' \,
       \ '[']', '72#B8' \, \ '/', '74#B8' \, \ '.',
    '75#B8' \
       \ #.

#DF hollerith-constant-value (unit)

     "{unit #IS <hollerith-constant> #AND unit #EQ
     current-executable-unit}"

     => hollerith-value-for ( #STRING-OF-TERMINALS-OF
        (hollerith-text-of (unit))) #.


#DF hollerith-text-of(unit)

     "{ unit #IS <hollerith-constant> #AND unit #EQ
     current-executable-unit }"

     => #SEG 4 #OF unit #.


#DF hollerith-value-for (str)

     "{str #IS <sign-string>}"

     => #NIL #IF #LENGTH (str) =0;

     => hollerith-for ( #FIRST-CHARACTER-IN (str) ) #CW
        hollerith-value-for (all-but-first-character-in
        (str)) #OTHERWISE #.


#DF hollerith-for (byte)

     "{byte #IS <sign>}"

     => ($six-bit-binary-representation-of-hollerith-for
        (byte)$) with-filler-bits-added #.


#DF six-bit-binary-representation-of-hollerith-for (byte)
```

```
    "{byte #IS <sign>}"

    => #PREFIX-OF-FIRST '#B2'#IN ( #CONVERT 2 (2
       #TH-ELEMENT-IN ( #FIRST map-pair #IN
       hollerith-map-pair-sequence #SUCH-THAT (
       #FIRST-ELEMENT-IN map-pair #EQW byte)))) #.


#DF hollerith-map-pair-sequence

    => implementation-standard-hollerith-map-pair-sequence
       #CS implementation-hollerith-map-pair-sequence #.


#DF status-constant-value (scon)

    "{scon #IS <status-constant>}"

    => ($ position-of(scon,"in"
       declaration-pertaining-to(scon) ) $)
       converted-to-implementation-form#.


#DF position-of (scon, "in" s-decl)

    => #ORDPOSIT (#FIRST stat-con #IN
       list-of-status-constants-in (s-decl) #SUCH-THAT
       (stat-con#EQW scon)) #IN list-of-status-constants-in
       (s-decl)#.


#DF list-of-status-constants-in (s-decl)

    => #SEQUENCE-OF <status-constant> #IN s-decl #.


#DF declaration-pertaining-to (scon)

    "{scon #IS <status-constant>}"

    => receiving-variable-declaration
       (assignment-containing (scon)) #IF ($scon$)
       is-assigned-status-constant;

    => compared-variable-declaration
       (status-relation-containing (scon)) #IF ($scon$)
       is-comparison-status-constant #.
```

```
#DF is-assigned-status-constant (scon)

    "{scon #IS <status-constant>}"

    => #FALSE #IF #NOT #THERE-EXISTS start #IN
       #SEQUENCE-OF-ANCESTORS-OF (scon) #SUCH-THAT (start
       #IS <assignment-statement>);

    => scon #EQ status-constant-of
       (rhs(assignment-containing(scon))) #IF
       ($assignment-containing(scon)$)
       is-atomic-assignment;

    => #FALSE #OTHERWISE #.


#DF is-atomic-assignment(stmt)

    "{stmt #IS <assignment-statement>}"

    => #TRUE #IFF stmt #IS #CASE 3 #OF
       <assignment-statement>#.


#DF assignment-containing (scon)

    "{scon #IS <status-constant>}"

    => #LAST stmt #IN (#SEQUENCE-OF-ANCESTORS-OF scon)
       #SUCH-THAT (stmt #IS <assignment-statement>) #.


#DF rhs (stmt)

    "{stmt #IS <assignment-statement>}"

    => #SEG 5 #OF stmt #.


#DF status-constant-of (atom-form)

    "{atom-form #IS <atomic-formula>}"

    => #SEG 1 #OF atom-form #.


#DF receiving-variable-declaration (stmt)
```

```
     "{stmt #IS <assignment-statement>}"

     => declaration-for (operand1-of (stmt)) #.

#DF status-relation-containing (scon)

     "{($scon$) is-comparison-status-constant}"

     => #PARENT-NODE (scon) #.

#DF is-comparison-status-constant (scon)

     "{scon #IS <status-constant>}"

     => #PARENT-NODE (scon) #IS <status-relation> #.

#DF compared-variable-declaration (srel)

     "{srel #IS <status-relation>}"

     => declaration-for (operand1-of (srel)) #.

#DF zero-constant-value (unit)

     "{unit #IS <zero>}"
     => '0' #.

#DF floating-latest-value (operand)

     "{($operand$) is-operation-or-primitive-operand}"

     => latest-value (operand) #IF type (operand) #EQW
        'floating';

     => ($ latest-value (operand), "with" attributes
        (operand) $) converted-fixed-to-floating #IF type
        (operand) #EQW 'fixed';

     => ($ latest-value (operand), "with" attributes
        (operand) $) converted-integer-to-floating #IF type
        (operand) #EQW 'integer' #.
```

```
#DF converted-integer-to-floating (val, "with" attr)

    "{ ($val$) is-implementation-numeric-representation
    #AND ($attr$) is-attribute}"

    => ($($\($bits-in-floating-mantissa - 1$)
       converted-to-implementation-form,
       double-word-representation-of(val) \  $) normalized
       $) converted-to-implementation-floating-form #.


#DF converted-fixed-to-floating(val,"with"attr)

    "{($val$)is-implementation-numeric-representation #AND
    ($attr$)is-attribute}"

    => ($ ($ \ ($ bits-per-word - 1 -
       fraction-bits-from(attr)$)
       converted-to-implementation-form,
       double-word-representation-of(val) \ $) normalized
       $) converted-to-implementation-floating-form #.


#DF converted-floating-to-integer (val)

    "{($val$)is-implementation-numeric-representation}"

    => #LEFT bits-per-word #CHARACTERS-OF
       implementation-left-arithmetic-shift
       (floating-mantissa-from(extended-precision-form-of
       (val)), "by" ($floating-exponent-from
       (extended-precision-form-of(val))$)converted-to-standard-form
       - bits-in-floating-mantissa + 1) #.


#DF converted-fixed-to-integer (val, "with" attr)

    "{ ($val$) is-implementation-numeric-representation
    #AND ($attr$) are-attribute}"

    => implementation-integer-zero #IF
       integer-bits-from(attr) <= 0 ;

    => implementation-left-arithmetic-shift (val,"by" #NEG
       fraction-bits-from(attr)) #OTHERWISE #.
```

```
#DF type (unit)

    "{($unit$) is-operation-or-primitive-operand}"

    => numeric-binary-op-type (unit) #IF ($unit$)
       is-numeric-binary-operation;

    => numeric-unary-op-type (unit) #IF($unit$)
       is-numeric-unary-operation;

    => function-type (unit) #IF ($unit$)
       is-function-reference;

    => 'boolean' #IF ($unit$) is-boolean-operation #OR
       ($unit$) is-relational-operation;

    => variable-type (unit) #IF ($unit$) is-a-variable;

    => constant-type (unit) #IF ($unit$) is-a-constant #.


#DF numeric-binary-op-type (operation)

    "{($operation$) is-numeric-binary-operation}"

    => 'integer'  #IF ($operation$) is-special-index-sum;

    => 'floating' #IF either-operand-is-float-in
       (operation) #OR ($operation$)
       is-to-be-done-in-floating-form ;

    => 'fixed' #IF either-operand-is-fixed-in (operation)
       #OR operation #IS <quotient>;

    => 'integer' #OTHERWISE #.

    "For any fixed base and an integer 0 exponent JOCIT
    defines the type of the result to be integer. we shall
    accept AFM100-24 definition of the result, which is
    fixed."


#DF numeric-unary-op-type (operation)

    "{($operation$) is-numeric-unary-operation}"
```

```
    => type (operand1-of(operation)) #IF operation #IS
       <abs-function> #U <unary-minus> #U <unary-plus>;

    => 'integer' #IF operation #IS <nwdsen-function> #U
       <loc-function> #.


#DF function-type (fx)

    "{($fx$) is-function-reference}"

    => declaration-type (function-item-declaration-for
       (declaration-for (function-name-in-reference
       (fx))))#.


#DF function-name-in-reference(fx)

    "{fx #IS <function-call>}"

    => #SEG 1 #OF (#SEG 1 #OF fx) #.


#DF function-item-declaration-for (fdec)

    "{ fdec #IS <procedure-declaration> #U
    <subprogram-declaration>}"

    => #FIRST item-dec #IN potential-item-decs-for(fdec)
       #SUCH-THAT (name-declared-in(fdec) #EQW
       name-declared-in (item-dec))#.


#DF potential-item-decs-for (fdec)

    "{ fdec #IS <procedure-declaration> #U
    <subprogram-declaration>}"

    => #SEQUENCE-OF <simple-item-declaration> #IN
       optional-template-declarations-of (fdec) #IF fdec
       #IS <subprogram-declaration>;

    => #SEQUENCE-OF <simple-item-declaration> #IN
       optional-decl-list-of (procedure-head-of(fdec)) #CS
       #SEQUENCE-OF <simple-item-declaration> #IN
       procedure-body-of (fdec) #OTHERWISE #.
```

```
#DF optional-template-declarations-of (spdec)

    "{spdec #IS <subprogram-declaration>}"

    => #SEG 6 #OF spdec #.


#DF procedure-head-of(pdec)

    "{ pdec #IS <procedure-declaration>}"

    => #SEG 1 #OF pdec #.


#DF optional-decl-list-of (phead)

    "{phead #IS <procedure-head>}"

    => #SEG 8 #OF phead #.


#DF procedure-body-of (pdec)

    "{pdec #IS <procedure-declaration>}"

    => #SEG 3 #OF pdec #.


#DF variable-type (operand)

    "{ ($operand$) is-variable }"

    => special-variable-type-of (operand) #IF ($ operand $)
       is-a-special-variable ;

    => declaration-type (declaration-for (operand) )
       #OTHERWISE #.


#DF is-a-special-variable (var)

    "{($var$) is-a-variable}"

    => #TRUE #IFF var #IS <special-integer-variable> #U
       <special-fixed-variable> #U
       <special-literal-variable> #U
       <special-boolean-variable> #.
```

```
#DF special-variable-type-of (operand)

    "{($operand$)is-a-special-variable}"

    => 'integer' #IF operand #IS
       <special-integer-variable>;

    => 'fixed' #IF operand #IS <special-fixed-variable>;

    => 'boolean' #IF operand #IS
       <special-boolean-variable>;

    => type (object-variable-of(operand)) #IF operand #IS
       <special-literal-variable> #.


#DF object-variable-of (operand)

    "{ operand #IS <special-literal-variable> #U
    <special-fixed-variable> #U <special-boolean-variable>
    #OR ($operand$) is-bit-functional-modifier #OR
    ($operand$) is-char-functional-modifier }"

    => #FIRST nx #IN ( #SEQUENCE-OF-NODES-IN subject-of
       (operand)) #SUCH-THAT ( ($nx$) is-a-variable) #.


#DF subject-of (operand)

    => #SEG 11 #OF operand #IF operand #IS
       <special-literal-variable> #OR ($operand$)
       is-bit-functional-modifier;

    => #SEG 5 #OF operand #IF operand #IS
       <special-boolean-variable> #U
       <special-fixed-variable> #OR ($operand$)
       is-char-functional-modifier #.


#DF declaration-type (decl)

    "{($decl$) is-typed-data-declaration}"

    => type-implied-by (typed-item-description-of (decl))
       #.
```

```
#DF typed-item-description-of (decl)

     "{($decl$) is-typed-data-declaration}"

     => type-specifier-of (item-description-of (decl)) #.


#DF item-description-of (decl)

     "{($decl$) is-typed-data-declaration}"

     => #SEG 7 #OF decl #IF decl #IS <array-declaration> ;

     => #SEG 3 #OF decl #IF decl #IS <mode-directive>;

     => #SEG 5 #OF decl #IF decl #IS
        <simple-item-declaration> #U
        <ordinary-table-item-declaration> #U
        <string-item-declaration> #U
        <defined-entry-item-declaration> #.


#DF type-implied-by (desc)

     "{($ desc $)is-typed-item-description}"

     => 'integer' #IF desc #IS <integer-item-description>;

     => 'fixed' #IF desc #IS <fixed-item-description>;

     => 'floating' #IF desc #IS <floating-item-description>;

     => 'hollerith' #IF desc #IS
        <hollerith-item-description>;

     => 'status' #IF desc #IS <status-item-description>;

     => 'boolean' #IF desc #IS <boolean-item-description>;

     => 'transmission-code' #IF desc #IS
        <transmission-code-item-description> #.


#DF constant-type (const)

     "{($const$) is-constant}"

     => 'integer' #IF const #IS <integer-constant>;
```

```
     => 'fixed' #IF const #IS <fixed-constant>;

     => 'floating' #IF const #IS <floating-constant>;

     => 'hollerith' #IF const #IS <hollerith-constant>;

     => 'status' #IF const #IS <status-constant>;

     => 'boolean' #IF const #IS <boolean-constant>;

     => 'transmission-code' #IF const #IS
        <transmission-code-constant>;

     => 'octal' #IF const #IS <octal-constant>;

     => 'zero-type' #IF const #IS <zero> #.

#DF either-operand-is-float-in (operation)

     "{($operation$) is-numeric-binary-operation}"

     => #TRUE #IFF type (operand1-of(operation)) #EQW
        'floating' #OR type (operand2-of(operation)) #EQW
        'floating' #.

#DF is-to-be-done-in-floating-form (operation)

     => #FALSE #IF operation #IS-NOT <exponential> ;

     => #TRUE #IF type (operand1-of (operation)) #EQW
        'floating';

     => #TRUE #IF operand2-of (operation) #IS-NOT
        <integer-constant>;

     => #TRUE #IF ($operation$)
        result-would-be-too-large-unfloated;

     => #FALSE #OTHERWISE #.

#DF result-would-be-too-large-unfloated (operation)

     "( operation #IS <exponential> #AND type (operand1-of
     (operation)) #NEQW 'floating' #AND operand 2-of
```

```
    (operation) #IS <integer-constant>}"

    => #TRUE #IFF size-of-result-of (operand1-of
       (operation)) * ($ latest-value (operand2-of
       (operation))$) converted-to-standard-form
       >=bits-per-word #.

#DF either-operand-is-fixed-in (operation)

    "{($operation$) is-numeric-binary-operation}"

    => #TRUE #IFF type (operand1-of(operation)) #EQW
       'fixed' #OR type (operand2-of(operation)) #EQW
       'fixed' #.

#DF both-operands-are-integer-in (operation)

    "{($operation$) is-numeric-binary-operation}"

    => #TRUE #IFF type (operand1-of(operation)) #EQW
       'integer' #AND type (operand2-of(operation)) #EQW
       'integer' #.

#DF both-operands-are-fixed-in (operation)

    "{($operation$) is-numeric-binary-operation}"

    => #TRUE #IFF type (operand1-of(operation)) #EQW
       'fixed' #AND type (operand2-of(operation)) #EQW
       'fixed' #.

#DF has-one-fixed-and-one-integer-operand(operation)

    "{($operation$) is-numeric-binary-operation}"

    => #TRUE #IF type (operand1-of(operation)) #EQW
       'integer' #AND type (operand2-of(operation)) #EQW
       'fixed' ;

    => #TRUE #IF type (operand1-of(operation)) #EQW 'fixed'
       #AND type (operand2-of(operation)) #EQW 'integer' ;

    => #FALSE #OTHERWISE #.
```

```
#DF integer-operand-of (operation)

     "{($operation$) is-binary-numeric-operation #AND
     either-operand-is-integer-in (operation)}"

     => operand1-of(operation) #IF type
        (operand1-of(operation)) #EQW 'integer';

     => operand2-of(operation) #IF type
        (operand2-of(operation)) #EQW 'integer' #.


#DF fixed-point-operand-of (unit)

     "{($unit$) is-binary-numeric-operation #AND
     either-operand-is-fixed-in (unit)}"

     => operand1-of(unit) #IF type (operand1-of(unit)) #EQW
        'fixed';

     => operand2-of(unit) #IF type (operand2-of(unit)) #EQW
        'fixed' #.
```

```
#DF attributes (operand)

    "{ ($operand$) is-operation-or-primitive-operand}"

    => \ integer-bits-in-result-of (operand),
       fractional-bits-in-result-of (operand),
       minimal-bits-in-result-of (operand) \ #.

#DF integer-bits-from (attr)

    => #FIRST-ELEMENT-IN attr #.

#DF fraction-bits-from (attr)

    => 2 #TH-ELEMENT-IN attr #.

#DF minimal-bits-from (attr)

    => 3 #TH-ELEMENT-IN attr #.

#DF integer-bits-in-result-of (unit)

    "{ ($unit$) is-operation-or-primitive-operand #AND type
    (unit) #IS-IN \ 'fixed', 'integer' \ }"

    => minimum (unadjusted-integer-bits (unit),
       bits-per-word - 1 - fractional-bits-in-result-of
       (unit)) #.

#DF maximum (x,y)

    "{ x #IS #INTEGER #AND y #IS #INTEGER }"

    => x #IF x >=y ;

    => y #OTHERWISE #.

#DF minimum (x,y)

    "{ x #IS #INTEGER #AND y #IS #INTEGER }"
```

```
    => x #IF x <=y ;

    => y #OTHERWISE #.

#DF unadjusted-integer-bits (unit)

    "{ ($unit$) is-operation-or-primitive-operand #AND type
    (unit) #IS-IN\ 'fixed', 'integer'\ }"

    => integer-bits-for-binary-op (unit) #IF ($unit$)
       is-numeric-binary-operation;

    => integer-bits-for-unary-op (unit) #IF ($unit$)
       is-numeric-unary-operation;

    => integer-bits-for-variable (unit) #IF ($unit$)
       is-a-variable;

    => integer-bits-for-constant (unit) #IF ($unit$)
       is-a-constant;

    => integer-bits-for-function (unit) #IF ($unit$)
       is-function-reference #.

#DF integer-bits-for-binary-op (unit)

    "{ ($unit$) is-numeric-binary-operation #AND type
    (unit) #IS-IN \ 'fixed', 'integer' \ }"

    => 1 + maximum (integer-bits-in-result-of (operand1-of
       (unit)), integer-bits-in-result-of (operand2-of
       (unit))) #IF unit #IS <sum> #U <difference>;

    => integer-bits-in-result-of (operand1-of (unit)) +
       integer-bits-in-result-of (operand2-of (unit)) #IF
       unit #IS <product>;

    => integer-bits-for-quotient (unit) #IF unit #IS
       <quotient>;

    => integer-bits-for-exponential (unit) #IF unit #IS
       <exponential> #.

#DF integer-bits-for-quotient (unit)
```

```
    "{ unit #IS <quotient> #AND type (unit) #IS-IN \
    'fixed', 'integer' \}"

    => integer-bits-in-result-of (operand1-of (unit)) + 1 -
       minimal-bits-in-result-of (operand2-of (unit)) #IF
       both-operands-are-integer-in (unit);

    => integer-bits-in-result-of (operand1-of (unit)) +
       fractional-bits-in-result-of (operand2-of (unit)) +
       1 - minimal-bits-in-result-of (operand2-of (unit))
       #IF either-operand-is-fixed-in (unit) #.


#DF integer-bits-for-exponential (unit)

    "{ unit #IS <exponential> #AND type (unit) #IS-IN \
    'fixed', 'integer' \ }"

    => ($value (operand2-of (unit))$)
       converted-to-standard-form *
       integer-bits-in-result-of (operand1-of (unit)) #IF
       ($value (operand2-of (unit))$)
       converted-to-standard-form > 0 ;

    => 1 #OTHERWISE #.


#DF integer-bits-for-unary-op (unit)

    "{ ($unit$) is-numeric-unary-operation #AND type (unit)
    #IS-IN \ 'fixed', 'integer' \ }"

    => integer-bits-in-result-of (operand1-of (unit)) #IF
       unit #IS <abs-function> #U <unary-minus> #U
       <unary-plus>;

    => nwdsen-constant-size (unit) #IF unit #IS
       <nwdsen-function>;

    => implementation-location-value-size #IF unit #IS
       <loc-function> #.


#DF nwdsen-constant-size (unit)

    "{unit #IS <nwdsen-function>}"
```

```
     => #LENGTH ( #PREFIX-OF-FIRST '#B2' #IN (#CONVERT 2
        (number-of-words-per-entry-in (declaration-for
        (tabular-name-of(unit)))))) #.

#DF tabular-name-of (nfune)

     "{nfune #IS <nwdsen-function>}"

     => #SEG 5 #OF nfune #.

#DF integer-bits-for-variable (unit)

     "{($unit$)is-a-variable #AND type(unit) #IS-IN
     \'fixed','integer'\}"

     => special-variable-integer-bits (unit) #IF ($unit$)
        is-a-special-variable;

     => declaration-integer-bits (declaration-for (unit))
        #OTHERWISE #.

#DF special-variable-integer-bits(unit)

     "{unit #IS
     <special-fixed-variable>.u<special-integer-variable>}"

     => 0 #IF unit #IS <special-fixed-variable>;

     => implementation-integer-bits-in-loop-variable #IF
        ($unit$)is-loop-variable;

     => integer-bits-in-bit-functional-modifier(unit) #IF
        ($unit$)is-bit-functional-modifier;

     => bits-in-floating-exponent - 1 #IF
        ($unit$)is-char-functional-modifier;

     => implementation-max-pos-functional-modifier-size #IF
        ($unit$)is-pos-functional-modifier;

     => nent-size(unit) #IF ($unit$)
        is-nent-functional-modifier #.

#DF is-loop-variable(var)
```

```
     "{var #IS <special-integer-variable>}"

     => #TRUE #IFF var #IS #CASE 1 #OF
        <special-integer-variable>#.


#DF integer-bits-in-bit-functional-modifier(unit)

     "{($unit$)is-bit-functional-modifier}"

     => integer-bits-in-result-of(object-variable-of(unit))
        #.


#DF is-bit-functional-modifier(var)

     => #TRUE #IFF var #IS #CASE 2 #OF
        <special-integer-variable>#.


#DF is-char-functional-modifier(var)

     => #TRUE #IFF var #IS #CASE 3 #OF
        <special-integer-variable>#.


#DF is-pos-functional-modifier(var)

     => #TRUE #IFF var #IS #CASE 4 #OF
        <special-integer-variable>#.


#DF is-nent-functional-modifier(var)

     => #TRUE #IFF var #IS #CASE 5 #OF
        <special-integer-variable>#.


#DF nent-size(unit)

     "{ ($unit$)is-neat-functional-modifier}"

     => number-of-bits-in
        (table-size-specified-in(declaration-for
        (table-name-of(unit))))#.
```

```
#DF table-name-of(nent-mod)

     "{($nent-mod$)is-nent-functional-modifier}"

     => #SEG 1 #OF (#SEG 5 #OF nent-mod)#.

#DF table-size-specified-in(table-dec)

     "{table-dec #IS <ordinary-table-declaration> #U
     <defined-entry-table-declaration> #U
     <like-table-declaration>}"

     => #STRING-OF-TERMINALS-OF(#SEG 3 #OF
        (table-size-specification-of(table-dec)))#.

#DF table-size-specification-of(table-dec)

     "{table-dec #IS <ordinary-table-declaration> #U
     <defined-entry-table-declaration> #U
     <like-table-declaration>}"

     => #SEG 5 #OF table-dec #IF table-dec #IS
        <ordinary-table-declaration> #U
        <defined-entry-table-declaration>;

     => #SEG 1 #OF
        optional-table-size-specification-of(table-dec) #IF
        optional-table-size-specification-of(table-dec)
        #NEQW #NIL;

     => table-size-specification-of(pattern-table-decl-for
        (table-dec)) #IF table-dec #IS
        <like-table-declaration>#.

#DF optional-table-size-specification-of(table-dec)

     "{table-dec #IS <like-table-declaration>}"

     => #SEG 5 #OF table-dec #.

#DF declaration-integer-bits(decl)

     "{($decl$) is-typed-data-declaration}"
```

```
    => integer-bits-for-fixed-decl (decl) #IF
       declaration-type (decl) #EQW 'fixed';

    => integer-bits-for-integer-decl (decl) #IF
       declaration-type (decl) #EQW 'integer' #.


#DF integer-bits-for-integer-decl (decl)

    "{declaration-type (decl) #EQW 'integer'}"

    => integer-bits-for-unsigned-integer-decl (decl) #IF
       ($decl$) is-unsigned-declaration;

    => integer-bits-for-signed-integer-decl (decl)
       #OTHERWISE#.


#DF integer-bits-for-unsigned-integer-decl (decl)

    "{declaration-type (decl) #EQW 'integer' #AND ($decl$)
    is-unsigned-declaration}"

    => integer-bits-from-integer-item-description
       (typed-item-description-of(decl))#.


#DF integer-bits-for-signed-integer-decl (decl)

    "{ declaration-type (decl) #EQW 'integer' #AND (#NOT
    ($decl$) is-unsigned-declaration)}"

    => integer-bits-from-integer-item-description
       (typed-item-description-of (decl)) - 1 #.


#DF integer-bits-from-integer-item-description (iid)

    "{iid #IS <integer-item-description>}"

    => minimum (integer-bits-from-integer-specifier-of
       (iid), "and" integer-bits-from-value-range-of (iid))
       #IF ($iid$) has-a-value-range;

    => integer-bits-from-integer-specifier-of (iid)
       #OTHERWISE #.
```

```
#DF has-a-value-range (item-desc)

     "{item-desc #IS <integer-item-description>
     .u<fixed-item-description>}"

     => #TRUE #IFF optional-value-range-of (item-desc) #NEQW
        #NIL #.

#DF optional-value-range-of (item-desc)

     "{item-desc #IS <integer-item-description>
     ,u<fixed-item-description>}"

     => #SEG 3 #OF item-desc #.

#DF integer-bits-from-value-range-of (item-desc)

     "{item-desc #IS <integer-item-description> #U
     <fixed-item-description>}"

     =>integer-bits-in-result-of (range-high-value
        (optional-value-range-of (item-desc))) #.

#DF range-high-value (vrange)

     "{vrange #IS <optional-integer-value-range> + ge> #U
     <optional-fixed-value-range> #AND vrange #NEQW #NIL}"

     => #SEG 6 #OF vrange #IF vrange #IS
        <optional-integer-value-range>;

     =, #SEG 1 #OF (#SEG 6 #OF vrange) #OTHERWISE #.

#DF integer-bits-from-integer-specifier-of (iid)

     "{iid #IS <integer-item-description>}"

     => specified-size-of (integer-specifier-of (iid)) #.

#DF integer-specifier-of (iid)

     "{iid #IS <integer-item-description>}"
```

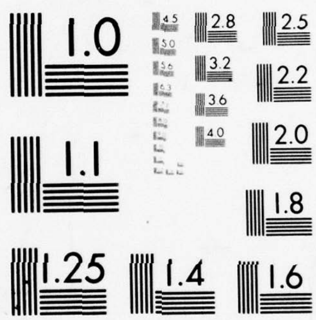
```
    => #SEG 1 #OF iid #.

#DF specified-size-of (spec)

    "{ spec #IS <integer-specifier> #U <fixed-specifier>}"

    => #STRING-OF-TERMINALS-OF (size-specifier-of (spec))
       #.

#DF size-specifier-of (spec)

    "{spec #IS <integer-specifier> #U <fixed-specifier>}"

    => #SEG 3 #OF spec #.

#DF integer-bits-for-fixed-decl (decl)

    "{declaration-type (decl) #EQW 'fixed'}"

    => integer-bits-for-unsigned-fixed-decl (decl) #IF
       ($decl$) is-unsigned-declaration;

    => integer-bits-for-signed-fixed-decl (decl)
       #OTHERWISE#.

#DF integer-bits-for-unsigned-fixed-decl (decl)

    "{declaration-type(decl) #EQW 'fixed' #AND ($decl$)
    is-unsigned-declaration}"

    => integer-bits-from-fixed-item-description
       (typed-item-description-of(decl))#.

#DF integer-bits-for-signed-fixed-decl(decl)

    "{declaration-type(decl) #EQW 'fixed' #AND(#NOT
    ($decl$) is-unsigned-declaration}"

    => integer-bits-from-fixed-item-description
       (typed-item-description-of(decl)) - 1 #.

#DF integer-bits-from-fixed-item-description(fid)
```

```
    "{fid #IS <fixed-item-description>}"

    => minimum(integer-bits-from-fixed-specifier-of(fid),
       "and" integer-bits-from-value-range-of(fid))#IF
       ($fid$)has-a-value-range;

    => integer-bits-from-fixed-specifier-of (fid)
       #OTHERWISE #.


#DF integer-bits-from-fixed-specifier-of (fid)

    "{ fid #IS <fixed-item-description>}"

    => specified-size-of (fixed-specifier-of (fid)) -
       specified-fraction-bits-of (fixed-specifier-of
       (fid)) #.


#DF fixed-specifier-of (fid)

    "{fid #IS <fixed-item-description>}"

    => #SEG 1 #OF fid #.


#DF specified-fraction-bits-of (fix-spec)

    "{fix-spec #IS <fixed-specifier>}"

    => #STRING-OF-TERMINALS-OF
       (fraction-bits-sign-of(fix-spec)) #CW
       #STRING-OF-TERMINALS-OF
       (fraction-bits-count-of(fix-spec)) #.


#DF fraction-bits-sign-of (fix-spec)

    "{fix-spec #IS <fixed-specifier>}"

    => #SEG 7 #OF fix-spec #.


#DF fraction-bits-count-of (fix-spec)

    "{fix-spec #IS <fixed-specifier>}"
```

```
    => #SEG 8 #OF fix-spec #.

#DF integer-bits-for-constant (unit)

    "{ unit #IS <integer-constant> #U <fixed-constant> }"

    => 1 - fractional-bits-for-constant (unit) #IF
       integer-bits-counted-in-constant (unit) +
       fractional-bits-for-constant (unit) <= 0;

    "since constant-value (unit) is zero"

    => integer-bits-counted-in-constant (unit) #OTHERWISE
       #.

#DF integer-bits-counted-in-constant (unit)

    "{ unit #IS <fixed-constant> #U <integer-constant> }"

    => number-of-bits-in (integer-portion-indicated-in
       (unit)) #.

#DF number-of-bits-in (val)

    "{ val #IS-IN #NATNOS }"

    => #LENGTH (#PREFIX-OF-FIRST '#B2' #IN ( #CONVERT 2
       (val))) #.

#DF integer-bits-for-function (unit)

    "{ ($unit$)is-function-reference #AND type (unit)
    #IS-IN \'fixed','integer'\}"

    =>
       declaration-integer-bits(function-item-declaration-for
       (declaration-for (function-name-in-reference
       (unit)))) #.

#DF fractional-bits-in-result-of (unit)

    "{ ($unit$) is-operation-or-primitive-operand #AND type
    (unit) #IS-IN \ 'fixed', 'integer' \ }"
```

AD-A049 474   TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CALIF   F/G 9/2
SEMANOL (76) SPECIFICATION OF JOVIAL (J3). VOLUME III.(U)
NOV 77 F C BELZ, I M GREEN   F30602-76-C-0238
UNCLASSIFIED   RADC-TR-77-365-VOL-3   NL

3 OF 4
AD
A049474


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

=> minimum (unadjusted-fractional-bits (unit), maximum (0, bits-per-word - 1 - unadjusted-integer-bits (unit))) #IF bits-per-word - 1 >= unadjusted-integer-bits (unit);

=> minimum (unadjusted-fractional-bits (unit), bits-per-word - 1 - unadjusted-integer-bits (unit)) #OTHERWISE #.

#DF unadjusted-fractional-bits (unit)

"{ ($unit$) is-operation-or-primitive-operand #AND type (unit) #IS-IN \ 'fixed', 'integer' \ }"

=> fractional-bits-for-binary-op (unit) #IF ($unit$) is-numeric-binary-operation;

=> fractional-bits-for-unary-op (unit) #IF ($unit$) is-numeric-unary-operation;

=> fractional-bits-for-variable (unit) #IF ($unit$) is-a-variable;

=> fractional-bits-for-constant (unit) #IF ($unit$) is-a-constant;

=> fractional-bits-for-function (unit) #IF ($unit$) is-function-reference #.

#DF fractional-bits-for-binary-op (unit)

"{ ($unit$) is-numeric-binary-operation #AND type (unit) #IS-IN \ 'fixed', 'integer' \}"

=> fractional-bits-for-sum-or-difference (unit) #IF unit #IS <sum> #U <difference>;

=> fractional-bits-for-product (unit) #IF unit #IS <product>;

=> fractional-bits-for-quotient (unit) #IF unit #IS <quotient>;

=> fractional-bits-for-exponential (unit) #IF unit #IS <exponential> #.

```
#DF fractional-bits-for-sum-or-difference (unit)

     "{ unit #IS <sum> #U <difference> #AND type (unit)
     #IS-IN \ 'fixed', 'integer' \ }"

     => 0 #IF type (unit) #EQW 'integer';

     => fractional-bits-in-result-of (operand1-of (unit))
        #IF fractional-bits-in-result-of (operand1-of
        (unit)) = fractional-bits-in-result-of (operand2-of
        (unit));

     => fractional-bits-in-result-of (fixed-point-operand-of
        (unit)) #IF ($unit$)
        has-one-fixed-and-one-integer-operand #AND
        fractional-bits-in-result-of (fixed-point-operand-of
        (unit)) >=0;

     => 1 + minimum (fractional-bits-in-result-of
        (operand1-of (unit)), fractional-bits-in-result-of
        (operand2-of (unit))) #OTHERWISE #.


#DF fractional-bits-for-product (unit)

     "{ unit #IS <product> #AND type (unit) #IS-IN \
     'fixed', 'integer' \}"

     => 0 #IF type (unit) #EQW 'integer';

     => fractional-bits-in-result-of (operand1-of (unit)) +
        fractional-bits-in-result-of (operand2-of (unit)) +
        1 - maximum (minimal-bits-in-result-of (operand1-of
        (unit)) , minimal-bits-in-result-of (operand2-of
        (unit))) #IF both-operands-are-fixed-in (unit);

     => fractional-bits-in-result-of (fixed-point-operand-of
        (unit)) + 1 + minimal-bits-in-result-of
        (integer-operand-of (unit)) #IF ($unit$)
        has-one-fixed-and-one-integer-operand #.


#DF fractional-bits-for-quotient (unit)

     "{ unit #IS <quotient> #AND type (unit) #IS-IN \
     'fixed', 'integer' \ }"
```

```
    => bits-per-word - 1 - integer-bits-in-result-of
       (operand1-of (unit)) #IF integer-bits-in-result-of
       (operand1-of (unit)) +
       preliminary-fractional-bits-for-quotient (unit) >
       bits-per-word;

    => preliminary-fractional-bits-for-quotient (unit)
       #OTHERWISE #.


#DF preliminary-fractional-bits-for-quotient (unit)

    "{ unit #IS <quotient> #AND type (unit) #IS-IN \
    'fixed', 'integer' \ }"

    => bits-per-word - 1 - maximum (unadjusted-integer-bits
       (unit), minimal-bits-in-result-of (operand1-of
       (unit))) #IF both-operands-are-integer-in (unit);

    => integer-bits-in-result-of (operand2-of (unit)) +
       fractional-bits-in-result-of (operand1-of (unit))
       #IF type (operand1-of (unit)) #EQW 'fixed';

    => 2 * integer-bits-in-result-of (operand2-of (unit)) +
       fractional-bits-in-result-of (operand2-of (unit)) -
       minimal-bits-in-result-of (operand1-of (unit)) #IF
       type (operand1-of (unit)) #EQW 'integer' #.


#DF fractional-bits-for-exponential (unit)

    "{ unit #IS <exponential> #AND type (unit) #IS-IN \
    'fixed', 'integer' \ }"

    => 0 #IF type (unit) #EQW 'integer' #OR
       #STRING-OF-TERMINALS-OF (operand2-of (unit)) = 0;

    => #STRING-OF-TERMINALS-OF (operand2-of (unit)) *
       (fractional-bits-in-result-of (operand1-of (unit)) -
       minimal-bits-in-result-of (operand1-of (unit)) + 1)
       + minimal-bits-in-result-of (operand1-of (unit)) - 1
       #OTHERWISE #.


#DF fractional-bits-for-unary-op (unit)

    "{ ($unit$) is-numeric-unary-operation #AND type (unit)
```

```
    #IS-IN \ 'fixed', 'integer' \ }"

    => fractional-bits-in-result-of (operand1-of (unit))
       #IF unit #IS <abs-function> #U <unary-minus> #U
       <unary-plus>;

    => 0 #IF unit #IS <nwdsen-function> #U <loc-function>
       #.

#DF fractional-bits-for-variable (unit)

    "{($unit$) is-a-variable #AND type (unit)
    #IS-IN\'fixed','integer'\}"

    => special-variable-fractional-bits(unit) #IF ($unit$)
       is-a-special-variable;

    =>
       declaration-fractional-bits(declaration-for(unit))#OTHERWISE#.

#DF special-variable-fractional-bits(unit)

    "{unit #IS <special-fixed-variable> #U
    <special-integer-variable>}"

    => 0 #IF unit #IS <special-integer-variable>;

    => bits-in-floating-mantissa - 1 #IF unit #IS
       <special-fixed-variable>#.

#DF declaration-fractional-bits (decl)

    "{($decl$)is-typed-data-declaration}"

    => 0 #IF declaration-type (decl) #EQW 'integer';

    => fractional-bits-from-fixed-item-description
       (typed-item-description-of(decl)) #IF
       declaration-type(decl) #EQW 'fixed' #.

#DF fractional-bits-from-fixed-item-description (fid)

    "{fid #IS <fixed-item-description>}"
```

```
    => minimum
       (fractional-bits-from-fixed-specifier-of(fid),"and"
       fractional-bits-from-value-range-of(fid)) #IF
       ($fid$) has-a-value-range;

    =>
       fractional-bits-from-fixed-specifier-of(fid)#OTHERWISE
       #.


#DF fractional-bits-from-fixed-specifier-of (fid)

    "{fid #IS <fixed-item-description>}"

    => specified-fraction-bits-of (fixed-specifier-of
       (fid)) #.


#DF fractional-bits-from-value-range-of (fid)

    "{fid #IS <fixed-item-description>}"

    => fractional-bits-in-result-of (range-high-value
       (optional-value-range-of (fid))) #.


#DF fractional-bits-for-constant (unit)

    "{ ($unit$) is-a-constant #AND type (unit) #IS-IN \
    'fixed', 'integer' \ }"

    => 0 #IF type (unit) #EQW 'integer';

    => #STRING-OF-TERMINALS-OF (fixed-constant-exponent-of
       (unit)) #IF type (unit) #EQW 'fixed' #.


#DF fractional-bits-for-function (unit)

    "{ ($unit$) is-function-reference #AND type (unit)
    #IS-IN \ 'fixed', 'integer' \ }"

    => declaration-fractional-bits
       (function-item-declaration-for (declaration-for
       (function-name-in-reference(unit))))#.


#DF minimal-bits-in-result-of (unit)
```

```
    "{ ($unit$) is-operation-or-primitive-operand #AND type
    (unit) #IS-IN\ 'fixed', 'integer' \ }"

    => minimum (unadjusted-minimal-bits (unit),
       bits-per-word - 1) #IF unadjusted-minimal-bits
       (unit) > 0;

    => 1 #OTHERWISE #.

#DF unadjusted-minimal-bits (unit)

    "{ ($unit$) is-operation-or-primitive-operand #AND type
    (unit) #IS-IN \ 'fixed', 'integer' \ }"

    => minimal-bits-for-binary-op (unit) #IF ($unit$)
       is-numeric-binary-operation;

    => minimal-bits-for-unary-op (unit) #IF ($unit$)
       is-numeric-unary-operation;

    => minimal-bits-for-variable (unit) #IF ($unit$)
       is-a-variable;

    => minimal-bits-for-constant (unit) #IF ($unit$)
       is-a-constant;

    => minimal-bits-for-function (unit) #IF ($unit$)
       is-function-reference #.

#DF minimal-bits-for-binary-op (unit)

    "{ ($unit$) is-numeric-binary-operation #AND type
    (unit) #IS-IN \ 'fixed', 'integer' \ }"

    => minimal-bits-for-sum-or-difference (unit) #IF unit
       #IS <sum> #U <difference>;

    => minimal-bits-for-product (unit) #IF unit #IS
       <product>;

    => minimal-bits-for-quotient (unit) #IF unit #IS
       <quotient>;

    => minimal-bits-for-exponential (unit) #IF unit #IS
       <exponential> #.
```

```
#DF minimal-bits-for-sum-or-difference (unit)

    "{ unit #IS <sum> #U <difference> #AND type (unit)
    #IS-IN \ 'fixed', 'integer' \ }"

    => fractional-bits-in-result-of (unit) + maximum
       (minimal-bits-in-result-of (operand1-of (unit)) -
       fractional-bits-in-result-of (operand1-of (unit)),
       minimal-bits-in-result-of (operand2-of (unit)) -
       fractional-bits-in-result-of (operand2-of (unit)))
       #IF ($unit$) is-unsigned;

    => 1 #OTHERWISE #.
```

"Unsigned variables and zero are included in case 1 of this DF. AFM 100-24 includes unsigned variables and zero, but they are not included in JOCIT."

```
#DF minimal-bits-for-product (unit)

    "{ unit #IS <product> #AND type (unit) #IS-IN \
    'fixed', 'integer' \}"

    => minimal-bits-in-result-of (operand1-of (unit)) +
       minimal-bits-in-result-of (operand2-of (unit)) - 1
       #IF both-operands-are-integer-in (unit);

    => minimum (minimal-bits-in-result-of (operand1-of
       (unit)), minimal-bits-in-result-of (operand2-of
       (unit))) #IF both-operands-are-fixed-in (unit);

    => minimal-bits-in-result-of (fixed-point-operand-of
       (unit)) #OTHERWISE #.

#DF minimal-bits-for-quotient (unit)

    "{ unit #IS <quotient> #AND type (unit) #IS-IN \
    'fixed', 'integer' \}"

    => maximum ( 1, minimal-bits-in-result-of (operand1-of
       (unit)) - minimal-bits-in-result-of (operand2-of
       (unit))) #.
```

```
#DF minimal-bits-for-exponential (unit)

    "{ unit #IS <exponential> #AND type (unit) #IS-IN \
    'fixed', 'integer #AND . not ($unit$)
    is-to-be-done-in-floating-form}"

    => #STRING-OF-TERMINALS-OF (operand2-of (unit)) *
       minimal-bits-in-result-of (operand1-of (unit)) -
       #STRING-OF-TERMINALS-OF (operand2-of (unit)) + 1 #IF
       type (unit) #EQW 'integer';

    => minimal-bits-in-result-of (operand1-of (unit)) #IF
       type (unit) #EQW 'fixed' #AND
       #STRING-OF-TERMINALS-OF (operand2-of (unit)) #NEQ 0;

    => 1 #OTHERWISE #.

#DF minimal-bits-for-unary-op (unit)

    "{ ($unit$) is-numeric-unary-operation #AND type (unit)
    #IS-IN \ 'fixed', 'integer' \ }"

    => minimal-bits-in-result-of (operand1-of (unit)) #IF
       unit #IS <abs-function> #U <unary-minus> #U
       <unary-plus>;

    => implementation-minimal-bits-in-nwdsen-function
       (unit) #IF unit #IS <nwdsen-function>;

    => implementation-minimal-bits-in-loc-function (unit)
       #IF unit #IS <loc-function> #.

#DF implementation-minimal-bits-in-nwdsen-function (unit)

    "{ unit #IS <nwdsen-function> #AND type (unit) #EQW
    'integer' }"

    => 1 #.

#DF implementation-minimal-bits-in-loc-function (unit)

    "{ unit #IS <loc-function> #AND type (unit) #EQW
    'integer' }"
```

=> 1 #.

#DF minimal-bits-for-variable (unit)

"{ ($unit$) is-a-variable #AND type (unit) #IS-IN \
'fixed','integer'\}"

=> special-variable-minimal-bits (unit) #IF ($unit$)
is-a-special-variable;

=> declaration-minimal-bits (declaration-for (unit))
#OTHERWISE #.

#DF special-variable-minimal-bits (unit)

"{ unit #IS <special-fixed-variable> #U
<special-integer-variable>}"

=> special-variable-fractional-bits (unit) #IF unit #IS
<special-fixed-variable>;

=> 1 #IF ($unit$) is-loop-variable #OR ($unit$)
is-pos-functional-modifier;

=> minimal-bits-in-bit-functional-modifier (unit) #IF
($unit$) is-bit-functional-modifier;

=> special-variable-integer-bits (unit) #IF ($unit$)
is-char-functional-modifier ;

=> minimal-bits-in-nent-functional-modifier (unit) #IF
($unit$) is-nent-functional-modifier #.

#DF minimal-bits-in-bit-functional-modifier (unit)

"{ ($unit$) is-bit-functional-modifier}"

=> minimal-bits-in-result-of (object-variable-of
(unit)) #.

#DF minimal-bits-in-nent-functional-modifier (unit)

"{ ($unit$) is-nent-functional-modifier)" =>
minimal-nent-size-specified-in (declaration-for

```
    (table-name-of (unit))) #.  #DF
    minimal-nent-size-specified-in (table-dec) "{ table-dec
    #IS <ordinary-table-declaration> #U
    <defined-entry-table-declaration> #U
    <like-table-declaration> }"

    => table-size-specified-in (table-dec) #IF
       ($table-dec$) is-a-rigid-table ;

    => 1 #OTHERWISE #.


#DF is-a-rigid-table (table-dec)

    "{ table-dec #IS <ordinary-table-declaration> #U
    <defined-entry-table-declaration> #U
    <like-table-declaration> }"

    => #TRUE #IFF #SEG 1 #OF table-size-specification-of
       (table-dec) #EQW 'r' #.


#DF declaration-minimal-bits (decl)

    "{ ($decl$) is-typed-data-declaration #AND
    declaration-type (decl) #IS-IN \'fixed','integer' \ }"

    => minimal-bits-from-item-description
       (typed-item-description-of (decl)) #.


#DF minimal-bits-from-item-description (item-desc)

    "{ item-desc #IS <integer-item-description> #U
    <fixed-item-description>?  + " =>
    minimal-bits-from-value-range-of (item-desc) #IF
    ($item-desc$) has-a-value-range;  => 1 #OTHERWISE #.
    #DF minimal-bits-from-value-range-of (item-desc) "{
    item-desc #IS <fixed-item-description> #U
    <integer-item-description> }"

    => minimal-bits-in-result-of (range-low-value
       (optional-value-range-of (item-desc))) #.


#DF range-low-value (vrange)

    "{ vrange #IS <optional-integer-range-value> #U
```

```
    <optional-fixed-value-range> #AND vrange #NEQW #NIL}"

    => #SEG 2 #OF vrange #IF vrange #IS
       <optional-integer-value-range>;

    => #SEG 1 #OF (#SEG 2 #OF vrange) #OTHERWISE #.

#DF minimal-bits-for-constant (unit)

    "{ ($unit$) is-a-constant #AND type (unit) #IS-IN \
    'fixed', 'integer' \ }"

    => integer-bits-in-result-of (unit) +
       fractional-bits-in-result-of (unit) #.

#DF minimal-bits-for-function (unit)

    "{ ($unit$) is-function-reference #AND type (unit)
    #IS-IN \ 'fixed', 'integer' \ }"

    => declaration-minimal-bits
       (function-item-declaration-for (declaration-for
       (function-name-in-reference (unit)))) #.

#DF is-unsigned (unit)

    "{ ($unit$) is-operation-or-primitive-operand}"

    => ( #TRUE #IFF ($unit$) is-unsigned-binary-op) #IF
       ($unit$) is-numeric-binary-operation;

    => ( #TRUE #IFF ($unit$) is-unsigned-unary-op) #IF
       ($unit$) is-numeric-unary-operation;

    => ( #TRUE #IFF ($unit$) is-unsigned-variable) #IF
       ($unit$) is-a-variable;

    => ( #TRUE #IFF ($unit$) is-unsigned-constant) #IF
       ($unit$) is-a-constant;

    => ( #TRUE #IFF ($unit$)
       is-unsigned-function-reference) #IF ($unit$)
       is-function-reference #.
```

```
#DF is-unsigned-binary-op (unit)

     "{ ($unit$) is-numeric-binary-operation #AND type
     (unit) #IS-IN \ 'fixed', 'integer' \ }"

     => #FALSE #IF unit #IS <difference>;

     => #TRUE #IFF ($operand1-of (unit)$) is-unsigned #AND
        ($operand2-of (unit)$) is-unsigned #OTHERWISE #.


#DF is-unsigned-unary-op (unit)

     "{ ($unit$) is-numeric-unary-operation #AND type (unit)
     #IS-IN \ 'fixed', 'integer' \ }"

     => #FALSE #IF unit #IS <unary-minus>;

     => #TRUE #IF unit #IS <loc-function> #U
        <nwdsen-function> ;

     => #TRUE #IF ($operand1-of (unit)$) is-unsigned;

     => #FALSE #OTHERWISE #.


#DF is-unsigned-variable (unit)

     "{ ($unit$) is-a-variable #AND type (unit) #IS-IN
     \'fixed','integer'\ }"

     => ($unit$) is-unsigned-special-variable #IF ($unit$)
        is-special-variable;

     => ($declaration-for (unit)$) is-unsigned-declaration
        #OTHERWISE #.


#DF is-special-variable (unit)

     "{ ($unit$) is-a-variable}"

     => #TRUE #IFF unit #IS <special-fixed-variable> #U
        <special-integer-variable> #.


#DF is-unsigned-special-variable (unit)
```

```
    "{ unit #IS <special-fixed-variable> #U
    <special-integer-variable>}"

    => #TRUE #IFF ($unit$) is-bit-functional-modifier #OR
       ($unit$) is-pos-functional-modifier #OR ($unit$)
       is-nent-functional-modifier #.


#DF is-unsigned-declaration (decl)

    "{ declaration-type (decl) #IS-IN \'fixed','integer'\}"

    => #TRUE #IFF signed-unsigned-designator-of
       (type-specifier-of (typed-item-description-of
       (decl))) #EQW 'u' #.


#DF type-specifier-of (item-desc)

    "{ item-desc #IS <fixed-item-description> #U
    <integer-item-description> }"

    => #SEG 1 #OF item-desc #.


#DF signed-unsigned-designator-of (specifier)

    "{ specifier #IS <integer-specifier> #U
    <fixed-specifier>}"

    => #SEG 5 #OF specifier #.


#DF is-unsigned-constant (unit)

    "{ ($unit$) is-a-constant #AND type (unit) #IS-IN \
    'fixed', 'integer' \ }"

    => #TRUE #.


#DF is-unsigned-function-reference (unit)

    "{ ($unit$) is-function-reference #AND type (unit)
    #IS-IN \ 'fixed', 'integer' \ }"

    => #TRUE #IFF ($function-item-declaration-for
       (declaration-for (function-name-in-reference
```

```
     (unit)))$) is-unsigned-declaration #.

#DF size-of-result-of (unit)

    "{ ($unit$) is-operation-or-primitive-operand}"

    => integer-bits-in-result-of (unit) +
       fractional-bits-in-result-of (unit) #.

#DF integer-attributes (operand)

    "{ ($ operand $) is-operation-or-primitive-operand }"

    => attributes (operand) #IF type (operand) #EQW
       'integer';

    => attributes-converted-fixed-to-integer(operand) #IF
       type(operand) #EQW 'fixed' ;

    => integer-attributes-of-converted-floating (operand)
       #IF type(operand) #EQW 'floating' #.

#DF attributes-converted-fixed-to-integer (operand)

    "{ ($ operand $) is-operation-or-primitive-operand }"

    => \ 0, 0, 1 \ #IF integer-bits-from (attributes
       (operand) ) <= 0 ;

    => \ integer-bits-from (attributes (operand) ),0,
       maximum (minimal-bits-from (attributes (operand) ) -
       fraction-bits-from (attributes (operand) ), 1 ) \
       #IF integer-bits-from (attributes (operand) ) > 0 #.

"This DF is implementation dependent."

#DF integer-attributes-of-converted-floating (operand)

    "{($ operand $) is-operation-or-primative-operand}"

    => \35,0,1\ #.
```

```
#DF index-list-comprising (index)

     "{ index #IS <destination-index> }"

     => #SEG 1 #OF index #.


#DF first-index-formula-in (ix-list)

     "{ ix-list #IS <index-list> }"

     => #SEG 1 #OF ix-list #.


#DF operand1-of (nx)

     => #FIRST-ELEMENT-IN
        sequence-of-operations-and-operands-in
        (first-operand-expression-of (nx) ) #.


#DF sequence-of-operations-and-operands-in (nx)

     => #SEQUENCE-OF-NODES ny #IN nx #SUCH-THAT ( ($ ny $)
        is-operation-or-primitive-operand ) #.


#DF operand2-of (nx)

     => #FIRST-ELEMENT-IN
        sequence-of-operations-and-operands-in
        (second-operand-expression-of (nx) ) #.


#DF first-operand-expression-of (nx)

     "{ ($ nx $) is-an-evaluated-formula #OR ($ nx $)
     is-operation-or-primitive-operand #AND #NOT ($ nx $)
     is-special-boolean-operation }"

     => first-operand-expression-of-numeric-binary-operation
        (nx) #IF ($ nx $) is-numeric-binary-operation;

     => first-operand-expression-of-numeric-unary-operation
        (nx) #IF ($ nx $) is-numeric-unary-operation;
```

```
    => first-operand-expression-of-boolean-operation (nx)
       #IF ($ nx $) is-boolean-operation;

    => first-operand-expression-of-relational-operation
       (nx) #IF ($ nx $) is-relational-operation;

    => nx #IF ($ nx $) is-an-evaluated-entity #.


#DF first-operand-expression-of-numeric-binary-operation
(nx)

    "{ ($ nx $) is-numeric-binary-operation }"

    => #SEG 1 #OF nx #.


#DF first-operand-expression-of-numeric-unary-operation (nx)

    "{ ($ nx $) is-numeric-unary-operation }"

    => #SEG 5 #OF nx #IF nx #IS #CASE 1 #OF <abs-function>
       #OR nx #IS <nwdsen-function> #U <loc-function>;

    => #SEG 3 #OF nx #IF nx #IS #CASE 2 #OF <abs-function>
       #OR nx #IS <unary-minus> #.


#DF first-operand-expression-of-boolean-operation (nx)

    "{ nx #IS <negation> }"

    => #SEG 3 #OF nx #.


#DF first-operand-expression-of-relational-operation (nx)

    "{ ($ nx $) is-relational-operation #AND nx #IS-NOT
    <chain-relation> }"

    => #SEG 1 #OF nx #.


#DF is-an-evaluated-entity(nx)

    "{nx #IS #NODE}"

    => #TRUE #IFF (($ nx $) is-an-evaluated-formula #OR ($
```

```
     nx $) is-an-evaluated-expression #OR ($ nx $)
     is-an-evaluated-atomic-formula) #.


#DF is-an-evaluated-atomic-formula (nx)

    "{ nx #IS #NODE }"

    => nx #IS <atomic-formula> #.


#DF is-an-evaluated-expression (nx)

    "{ nx #IS #NODE}"

    => nx #IS <boolean-expression> #U <numeric-expression>
       #.


#DF second-operand-expression-of (nx)

    "{ ($ nx $) is-binary-opeation #AND #NOT ($ nx $)
    is-special-boolean-operation }"

    => #SEG 5 #OF nx #.


#DF relation-constant-of (unit)

    "{ unit #IS <relation> #AND unit #EQ
    current-executable-unit }"

    => #RIGHT 2 #CHARACTERS-OF (#STRING-OF-TERMINALS-OF (
       #SEG 3 #OF unit ) ) #.


#DF formula-comprising (nx)

    "{ nx #IS <index-formula>}"

    => #SEG 1 #OF ( #SEG 1 #OF nx) #.


#DF exponent-part-of (nx)

    "{ ($nx$) has-an-exponent-part}"

    => #SEG 3 #OF nx #.
```

#DF floating-part-of (nx)

"{ nx #IS <fixed-constant>}"

=> #SEG 1 #OF nx #.

#DF has-a-fraction-part (nx)

"{ nx #IS #NODE}"

=> (#TRUE #IFF ($ simple-floating-constant-part-of (nx)$) has-a-fraction-part) #IF nx #IS <exponentiated-floating-constant>;

=> #TRUE #IFF nx #IS #CASE 2 #OF <simple-floating-constant> #OR nx #IS #CASE 3 #OF <simple-floating-constant> #OTHERWISE #.

#DF integer-part-of (nx)

"{ nx #IS <integer-constant> #OR (nx #IS <simple-floating-constant #U <exponentiated-floating-constant> #AND (nx) has-an-integer-part}"

=> integer-part-of (simple-floating-constant-part-of (nx)) #IF nx #IS <exponentiated-floating-constant>;

=> #SEG 1 #OF nx #OTHERWISE #.

#DF has-an-integer-part (nx)

"{ nx #IS #NODE}"

=> (#TRUE #IFF ($simple-floating-constant-part-of (nx)$) has-an-integer-part) #IF nx #IS <exponentiated-floating-constant>;

=> #TRUE #IFF nx #IS #CASE 1 #CF <simple-floating-constant> #OR nx #IS #CASE 2 #OF <simple-floating-constant> #OTHERWISE #.

```
#DF simple-floating-constant-part-of (nx)

     "{ nx #IS <exponentiated-floating-constant>}"

     => #SEG 1 #OF nx #.

#DF fixed-constant-exponent-of (fixed-const)

     "{ fixed-const #IS <fixed-constant>}"

     => #SEG 3 #OF fixed-const #.
```

```
#DF implementation-location-value-size

     => 18 #.

#DF implementation-max-pos-functional-modifier-size

     => bits-per-word - 1 #.

#DF implementation-integer-bits-in-loop-variable

     => bits-per-word - 1 #.

#DF bits-per-word

     "{on-return: bits-per-word >= bits-per-byte}"

     => 36 #.

#DF bits-per-byte

     "{on-return: bits-per-byte >= 6}"

     => 9 #.

     "The bits per byte are tacitly assumed to be greater or
     equal to six."

#DF implementation-integer-zero

     => ($ bits-per-word $) zeroes #.

#DF implementation-integer-add (x,y)

     "{ ($\x,y\$)
     are-implementation-numeric-representations}"

     => ($ ($x$) converted-to-standard-form + ($y$)
        converted-to-standard-form $)
        with-result-converted-to-implementation-form #.
```

```
#DF implementation-fixed-add (val-x, attr-x, val-y, attr-y,
        attr-r)

    "{($\val-x, val-y\$)
    are-implementation-numeric-representations #AND
    ($\attr-x, attr-y, attr-r\$) are-attributes}"

    => implementation-integer-add
       (implementation-left-arithmetic-shift (val-x,"by"
       fraction-bits-from (attr-r) - fraction-bits-from
       (attr-x)),"+" implementation-left-arithmetic-shift
       (val-y ,"by" fraction-bits-from (attr-r) -
       fraction-bits-from (attr-y))) #.


#DF implementation-integer-subtract (x,y)

    "{ ($\x,y\$)
    are-implementation-numeric-representations}"

    => ($ ($x$) converted-to-standard-form - ($y$)
       converted-to-standard-form $)
       with-result-converted-to-implementation-form #.


#DF implementation-fixed-subtract (val-x, attr-x, val-y,
        attr-y,attr-r)"{ ($\val-x, val-y\$)
        are-implementation-numeric-representations #AND
        ($\attr-x, attr-y, attr-r\$) are-attributes}"

    => implementation-integer-subtract
       (implementation-left-arithmetic-shift(val-x, "by"
       fraction-bits-from (attr-r) -
       fraction-bits-from(attr-x)) ,"-"
       implementation-left-arithmetic-shift (val-y ,"by"
       fraction-bits-from (attr-r) - fraction-bits-from
       (attr-y))) #.


#DF implementation-floating-subtract(val-x, "-" val-y)

    "{ ($ \val-x,val-y\ $)
    are-implementation-numeric-representations }"

    => implementation-floating-add (val-x, "+"
       implementation-negated-floating(val-y)) #.
```

```
#DF implementation-integer-product (x,y)

     "{ ($\x,y\$)
     are-implementation-numeric-representations}"

     => ($ ($x$) converted-to-standard-form * ($x$)
        converted-to-standard-form $)
        with-result-converted-to-implementation-form #.


#DF implementation-fixed-product
(val-x,attr-x,val-y,attr-y,attr-r)

     "{ ($\val-x, val-y\$)
     are-implementation-numeric-representations #AND
     ($\attr-x, attr-y, attr-r\$) are-attributes}"

     => ($ implementation-left-arithmetic-shift ( ($
        ($val-x$) converted-to-standard-form * ($val-y$)
        converted-to-standard-form $)
        with-result-converted-to-implementation-dp-form,
        "by" fraction-bits-from (attr-r) -
        fraction-bits-from (attr-x) - fraction-bits-from
        (attr-y) ) $) conformed-to-implementation-word-size
        #.


#DF implementation-integer-quotient (x,"by"y)

     "{ ($x$) is-implementation-double-word-numeric-rep #AND
     ($y$) is-implementation-numeric-representation}"

     => ($ ($x$)converted-to-standard-form /
        ($y$)converted-to-standard-form $)
        with-result-converted-to-implementation-form #.


#DF implementation-fixed-quotient (val-x, attr-x, val-y,
        attr-y,attr-r)"{ ($\val-x, val-y\$)
        are-implementation-numeric-representations #AND
        ($\attr-x, attr-y, attr-r\$) are attributes}"

     => implementation-integer-quotient
        (implementation-left-arithmetic-shift (val-x, "by"
        fraction-bits-from (attr-r) + fraction-bits-from
        (attr-y) - fraction-bits-from(attr-x) ) , "/" val-y)
        #.
```

```
#DF implementation-floating-compare (val-x, val-y)
    "{ ($\val-x, val-y\$)
    are-implementation-numeric-representations}"

    => relation-with-floating-zero-of
       (implementation-floating-subtract (val-x, val-y)) #.


#DF implementation-integer-and-fixed-point-compare
       (val-x, "with" attr-x, val-y, "with" attr-y)

    "{ ($\val-x, val-y\$)
    are-implementation-numeric-representations #AND
    ($\val-x,val-y\$) are-attributes}"
    => relation-with-integer-zero-of
       (implementation-fixed-subtract (val-x, "with"
       attr-x, val-y, "with" attr-y,
       result-attributes-for-difference-of (attributes
       (val-x),

    "and" attributes (val-y))))
       #.


#DF relation-with-floating-zero-of (val)
    "{ ($val$) is-implementation-floating-representation}"

    => '=' #IF ($val$) is-zero-floating-representation;

    => '>' #IF ($val$) is-positive-floating-representation;

    => '<' #IF ($val$) is-negative-floating-representation
       #.


#DF is-zero-floating-representation (val)
    "{ ($val$) is-implementation-floating-representation}"

    => val #EQW implementation-floating-zero #.


#DF is-positive-floating-representation(val)
```

```
     "{($val$) is-implementation-floating-representation}"

     => '0' #EQW sign-bit(floating-mantissa-of(val)) #.

#DF is-negative-floating-representation(val)

     "{($val$) is-implementation-floating-representation}"

     => '1' #EQW sign-bit(floating-mantissa-of(val)) #.

#DF relation-with-integer-zero-of (val)

     "{ ($val$) is-implementation-numeric-representation}"

     => '=' #IF val #EQW implementation-integer-zero ;

     => '>' #IF #FIRST-CHARACTER-IN (val) #EQW '0';

     => '<' #IF #FIRST-CHARACTER-IN (val) #EQW '1' #.

#DF result-attributes-for-difference-of (x, "and" y)

     "{ ($\x,y\$) are-attributes}"

     => \integer-bits-for-difference-of (x, "and" y),
        fractional-bits-for-difference-of (x, "and" y),
        minimal-bits-for-difference-of (x, "and" y)\ #.

"The relations used here for the result attributes of a
difference used in relations are the same as those used for
the difference operator."

#DF integer-bits-for-difference-of (x, "and" y)

     "{ ($\x,y\$) are-attributes}"

     => minimum (unadjusted-integer-bits-for-difference-of
        (x, "and" y), bits-per-word - 1 -
        fractional-bits-for-difference-of (x,

     "and" y))
        #.
```

```
#DF unadjusted-integer-bits-for-difference-of (x, "and" y)

    "{ ($\x, y\$) are-attributes}"

    => 1 + maximum (integer-bits-from (x),
       integer-bits-from (y)) #.


#DF fractional-bits-for-difference-of (x, "and" y)

    "{ ($\x,y\$) are-attributes}"

    => minimum
       (unadjusted-fractional-bits-for-difference-of (x,
       "and" y), maximum (0, bits-per-word - 1 -
       unadjusted-integer-bits-for-difference-of (x, "and"
       y))) #IF bits-per-word - 1 >=
       unadjusted-integer-bits-for-difference-of (x, "and"
       y);

    => minimum
       (unadjusted-fractional-bits-for-difference-of (x,
       "and" y), bits-per-word - 1 -
       unadjusted-integer-bits-for-difference-of (x, "and"
       y)) #OTHERWISE #.


#DF unadjusted-fractional-bits-for-difference-of (x, "and"
y)

    "{ ($\x,y\$) are-attributes}"

    => fraction-bits-from (x) #IF fraction-bits-from (x) =
       fraction-bits-from (y) #OR (fraction-bits-from (y) =
       0 #AND fraction-bits-from (x) >= 0) ;

    => fraction-bits-from (y) #IF fraction-bits-from (x) =0
       #AND fraction-bits-from (y) >= 0 ;

    => 1 + minimum (fraction-bits-from (x),
       fraction-bits-from (y)) #OTHERWISE #.


#DF minimal-bits-for-difference-of (x, "and" y)

    "{ ($\x, y\$) are-attributes}"
```

```
     => maximum (1, bits-per-word - 1) #.

#DF implementation-left-arithmetic-shift (val, "by" count)

     "{ ($val$) is-string-of-ones-and-zeroes #AND count #IS
     #INTEGER }"

     => val #IF count = 0;

     => implementation-right-arithmetic-shift (val, "by"
        #NEG count) #IF count < 0;

     => sign-bit (val) #CW (#RIGHT (#LENGTH(val) - 1)
        #CHARACTERS-OF (magnitude-bits (val) #CW
        ($count$)zeroes)) #OTHERWISE #.

#DF implementation-right-arithmetic-shift (val,"by" count)

     "{ ($val$) is-string-of-ones-and-zeroes #AND count #IS
     #INTEGER }"

     => '1' #CW (#LEFT (#LENGTH(val) - 1) #CHARACTERS-OF
        (($count$) ones #CW magnitude-bits (val))) #IF
        sign-bit (val) #EQW '1';

     => '0' #CW (#LEFT (#LENGTH(val) - 1) #CHARACTERS-OF
        (($count$) zeroes #CW magnitude-bits (val)))
        #OTHERWISE #.

#DF sign-bit(val)

     "{ ($val$) is-string-of-ones-and-zeroes }"

     => #FIRST-CHARACTER-IN (val) #.

#DF magnitude-bits(val)

     "{ ($val$) is-string-of-ones-and-zeroes }"

     =>  #SUBSTRING-OF-CHARACTERS 2 #TO #LENGTH(val) #OF val
        #.
```

```
#DF converted-to-standard-form (val)

     "{ ($val$) is-string-of-ones-and-zeroes }"

     => standard-sign (val) #CW standard-magnitude (val) #CW
        '#B2'#.

#DF standard-sign (val)

     "{ ($val$) is-string-of-ones-and-zeroes }"

     => '-' #IF #FIRST-CHARACTER-IN (val) #EQW '1';

     => #NIL #OTHERWISE #.

#DF standard-magnitude (val)

     "{ ($val$) is-string-of-ones-and-zeroes }"

     => ($val$) with-leftmost-zeroes-suppressed #IF
        #FIRST-CHARACTER-IN (val) #EQW '0';

     => ($ ($ ($ val $) decremented-by-one $) complemented
        $) with-leftmost-zeroes-suppressed #OTHERWISE #.

#DF decremented-by-one (val)

     "{ ($val$) is-implementation-numeric-representation}"

     => #RIGHT ( #LENGTH (val)) #CHARACTERS-OF (
        #PREFIX-OF-FIRST '#B2'#IN ('0'#CW ((val #CW '#B2') -
        1#B2))) #.

#DF complemented (val)

     "{ ($val$) is-string-of-ones-and-zeroes }"

     => #PREFIX-OF-FIRST '#BITS'#IN ( (val #CW '#BITS')
        #BXOR (($#LENGTH(val)$)ones #CW '#BITS') ) #.

#DF with-leftmost-zeroes-suppressed (x)

     "{($x$) is-implementation-numeric-representation}"
```

```
     => '0' #IF x = 0;

     => #SUBSTRING-OF-CHARACTERS
        (pos-of-first-non-zero-char-in (x)) #TO #LENGTH (x)
        #OF x #OTHERWISE #.


#DF pos-of-first-non-zero-char-in (x)

     "{ ($x$) is-implementation-numeric-representation}"

     => #FIRST char-pos : 1 <= char-pos <= #LENGTH(x)
        #SUCH-THAT (char-pos #TH-CHARACTER-IN x #NEQW '0')
        #.


#DF with-result-converted-to-implementation-form (sem-const)

     "{ ($sem-const$) is-semanol-base-2-integer-constant}"

     => ($ #PREFIX-OF-FIRST '#B2'#IN sem-const $)
        conformed-to-implementation-word-size #IF
        #FIRST-CHARACTER-IN (sem-const) #NEQW '-';

     => ($($($ word-between ('-', "and" '#B2', "in"
        sem-const)$) conformed-to-implementation-word-size
        $) complemented $) incremented-by-one #OTHERWISE #.


#DF word-between(string1, "and" string2, "in" sem-const)

     "{ ($\string1,string2,sem-const\$) are-semanol-strings
     }"

     => #PREFIX-OF-FIRST string2 #IN (#SUFFIX-OF-FIRST
        string1 #IN sem-const) #.


#DF conformed-to-implementation-word-size (val)

     "{ ($val$) is-string-of-ones-and-zeroes}"

     => #RIGHT bits-per-word #CHARACTERS-OF (($
        bits-per-word$) zeroes #CW val) #.


#DF incremented-by-one (val)
```

```
     "{ ($val$) is-string-of-ones-and-zeroes }"

     => #RIGHT ( #LENGTH (val)) #CHARACTERS-OF (
        #PREFIX-OF-FIRST '#B2' #IN ( (val #CW '#B2') +
        1#B2)) #.


#DF with-result-converted-to-implementation-dp-form
(sem-const)

     "{ ($sem-const$) is-semanol-base-2-integer-constant}"

     => ($ #PREFIX-OF-FIRST '#B2'#IN sem-const $)
        conformed-to-implementation-dp-word-size #IF
        #FIRST-CHARACTER-IN (sem-const) #NEQW '-';

     => ($($($ word-between ('-', "and" '#B2', "in"
        sem-const)$)
        conformed-to-implementation-dp-word-size $)
        complemented $) incremented-by-one #OTHERWISE #.


#DF conformed-to-implementation-dp-word-size (val)

     "{ ($val$) is-string-of-ones-and-zeroes}"

     => #RIGHT (2 * bits-per-word) #CHARACTERS-OF ( ($ (2 *
        bits-per-word) $) zeroes #CW val) #.


#DF fractional-binary-representation-of (d, "to"
     e "bits-of-precision")

     "{ e >=0 #AND d #IS-IN #NAT-NOS}"

     => #NIL #IF e <=0;

     => next-binary-digit-from (d) #CW
        fractional-binary-representation-of
        (binary-fractional-residue (d), "to" e - 1 "bits")
        #OTHERWISE #.


#DF next-binary-digit-from (d)

     "{ d #IS-IN #NAT-NOS}"
```

```
     => '0' #IF #FIRST-CHARACTER-IN (d) < 5;

     => '1' #OTHERWISE #.


#DF binary-fractional-residue (d)

     "{ d #IS-IN #NAT-NOS}"

     => #RIGHT ( #LENGTH (d)) #CHARACTERS-OF ( ($ #LENGTH(d)
        $) zeroes #CW (2*d) ) #.


#DF implementation-standard-hollerith-map-pair-sequence

     => \
           \#SPACE, '20#B8' \, \ 'A', '21#B8' \, \ 'B',
        '22#B8' \,
           \ 'C', '23#B8' \, \ 'D', '24#B8' \, \ 'E',
        '25#B8' \,
           \ 'F', '26#B8' \, \ 'G', '27#B8' \, \ 'H',
        '30#B8' \,
           \ 'I', '31#B8' \, \ 'J', '41#B8' \, \ 'K',
        '42#B8' \,
           \ 'L', '43#B8' \, \ 'M', '44#B8' \, \ 'N',
        '45#B8' \,
           \ 'O', '46#B8' \, \ 'P', '47#B8' \, \ 'Q',
        '50#B8' \,
           \ 'R', '51#B8' \, \ 'S', '62#B8' \, \ 'T',
        '63#B8' \,
           \ 'U', '64#B8' \, \ 'V', '65#B8' \, \ 'W',
        '66#B8' \,
           \ 'X', '67#B8' \, \ 'Y', '70#B8' \, \ 'Z',
        '71#B8' \,
           \ ')', '55#B8' \, \ '-', '52#B8' \, \ '+',
        '60#B8' \,
           \ '=', '75#B8' \, \ '$', '53#B8' \, \ '*',
        '54#B8' \,
           \ '(', '35#B8' \, \ ',', '73#B8' \, \ '0',
        '00#B8' \,
           \ '1', '01#B8' \, \ '2', '02#B8' \, \ '3',
        '03#B8' \,
           \ '4', '04#B8' \, \ '5', '05#B8' \, \ '6',
        '06#B8' \,
           \ '7', '07#B8' \, \ '8', '10#B8' \, \ '9',
        '11#B8' \,
           \ '[']', '57#B8' \, \ '/', '61#B8' \, \ '.',
        '33#B8' \
```

```
          \ #.


#DF implementation-hollerith-map-pair-sequence

     => \
          \ '[[]', '12#B8' \, \ ']', '34#B8' \,
          \ '<', '36#B8' \, \ '>', '16#B8' \,
          \ ':', '15#B8' \, \ ';', '56#B8' \,
          \ '?', '17#B8' \, \ '!', '77#B8' \,
          \ '%', '74#B8' \, \ '"', '76#B8' \,
          \ '@', '14#B8' \, \ '.', '13#B8' \,
          \ '&', '32#B8' \, \ '\', '37#B8' \,
          \ '^', '40#B8' \, \ '_', '72#B8' \
          \ #.


#DF implementation-floating-add(val-x,"+"val-y)

     "{($\val-x,val-y\$)
     are-implementation-numeric-representation}"

     => ($normalized-floating-add-process
        (extended-precision-form-of(val-x),"+"
        adjusted-extended-precision-form-of
        (val-y,"using"val-x))$)
        converted-to-implementation-floating-form #IF
        exponent-difference-of(val-x,"and"val-y) >=0;

     => implementation-floating-add(val-y,"+"val-x)
        #OTHERWISE #.


#DF normalized-floating-add-process (impl-val-x, "+"
impl-val-y)

     "{($\impl-val-x,impl-val-y\$)are-extended-precision-forms}"

     => ($ \ floating-exponent-from(impl-val-x),
        implementation-double-word-add
        (floating-mantissa-from(impl-val-x),"+"
        floating-mantissa-from(impl-val-y))\$) normalized #.


#DF extended-precision-form-of(val)

     "{ ($val$)is-implementation-numeric-representation}"
```

```
     => \ single-word-representation-of
        (floating-exponent-of(val)),
        double-word-representation-of
        (floating-mantissa-of(val))\ #.


#DF adjusted-extended-precision-form-of(val-y,"using"val-x)

     "{($\val-x,val-y\$)
     are-implementation-numeric-representation}"

     => \single-word-representation-of
        (floating-exponent-of(val-x)),
        implementation-right-arithmetic-shift
        (double-word-representation-of
        (floating-mantissa-of(val-y)),"by"
        exponent-difference-of(val-x,"and"val-y) ) \ #.


#DF exponent-difference-of(val-x,"and"val-y)

     "{($\val-x,val-y\$)
     are-implementation-numeric-representations}"

     => ($floating-exponent-from(extended-precision-form-of
        (val-x))$)converted-to-standard-form -
        ($floating-exponent-from(extended-precision-form-of
        (val-y)) $) converted-to-standard-form #.


#DF floating-exponent-from(impl-val)

     "{($impl-val$)is-extended-precision-form}"

     => #FIRST-ELEMENT-IN impl-val #.


#DF floating-mantissa-from(impl-val)

     "{($impl-val$)is-extended-precision-form}"

     => #LAST-ELEMENT-IN impl-val #.


#DF implementation-double-word-add(dpval-x,"and"dpval-y)

     "{($\dpval-x,dpval-y\$)
     are-implementation-double-word-numeric-reps}"
```

```
    => ($($dpval-x$)converted-to-standard-form +
       ($dpval-y$)converted-to-standard-form$)
       with-result-converted-to-implementation-dp-form #.


#DF normalized(impl-val)

    "{($impl-val$)is-extended-precision-form}"

    => \implementation-integer-subtract
       (floating-exponent-from(impl-val),"and"
       ($normalizing-left-shift-for (
       floating-mantissa-from (impl-val)) $)
       converted-to-implementation-form),
       implementation-left-arithmetic-shift
       (floating-mantissa-from(impl-val),"by"
       normalizing-left-shift-for
       (floating-mantissa-from(impl-val) ) ) \ #.


#DF converted-to-implementation-floating-form(impl-val)

    "{($impl-val$)is-extended-precision-form}"

    => implementation-floating-zero #IF
       floating-mantissa-from(impl-val)=0;

    => #SUBSTRING-OF-CHARACTERS (1 + bits-per-word -
       bits-in-floating-exponent) #TO bits-per-word #OF
       floating-exponent-from(impl-val) #CW (
       #SUBSTRING-OF-CHARACTERS ( 1 +
       bits-in-floating-exponent ) #TO bits-per-word #OF
       floating-mantissa-from (impl-val) ) #OTHERWISE #.


#DF single-word-representation-of(val)

    "{($val$)is-string-of-ones-and-zeroes}"

    => #RIGHT bits-per-word
       #CHARACTERS-OF(sign-extension-for(val) #CW val) #.


#DF floating-exponent-of(val)

    "{($val$)is-implementation-numeric-representation}"
```

```
    => #LEFT bits-in-floating-exponent #CHARACTERS-OF val
       #.

#DF double-word-representation-of(val)

    "{($val$)is-string-of-ones-and-zeroes}"

    => #RIGHT (2*bits-per-word) #CHARACTERS-OF
       (sign-extension-for(val)#CW val #CW
       ($bits-per-word$)zeroes)#.

#DF floating-mantissa-of(val)

    "{($val$)is-implementation-numeric-representation}"

    => #RIGHT bits-in-floating-mantissa #CHARACTERS-OF val
       #.

#DF normalizing-left-shift-for(dpval)

    "{($dpval$)is-implementation-double-word-numeric-rep}"

    => 0 #IF dpval=0;

    => (#SUBSTRING-POSIT-OF '1' #IN magnitude-bits(dpval))
       - (1+bits-in-floating-exponent) #IF
       sign-bit(dpval)#EQW '0' ;

    => (#SUBSTRING-POSIT-OF '0' #IN magnitude-bits(dpval))
       - (1 + bits-in-floating-exponent) #OTHERWISE #.

#DF converted-to-implementation-form(sem-const)

    "{($sem-const$)is-semanol-decimal-integer-constant}"

    => ($#CONVERT 2 (sem-const) $)
       with-result-converted-to-implementation-form #.

#DF implementation-floating-zero

    "{on-return:
    length(implementation-floating-zero)=bits-per-word}"
```

```
    => '1' #CW ($6$)zeroes #CW ($28$)zeroes #.


#DF bits-in-floating-exponent

    "{on-return:bits-in-floating-exponent >=8 #AND
    bits-in-floating-exponent bits-per-word}"

    => 8 #.


#DF bits-in-floating-mantissa

    "{on-return: bits-in-floating-mantissa>=28 #AND
    bits-in-floating-mantissa\ bits-per-word}"

    => 28 #.


#DF sign-extension-for(val)

    "{($val$)is-string-of-ones-and-zeroes}"

    => ($bits-per-word$)ones #IF sign-bit(val)#EQW'1';

    => ($bits-per-word$) zeroes #OTHERWISE #.


#DF implementation-negated-floating(val)

    "{($val$)is-implementation-numeric-representation}"

    => ($ \ floating-exponent-from
        (extended-precision-form-of(val)),
        implementation-negated (floating-mantissa-from
        (extended-precision-form-of(val)))\$)
        converted-to-implementation-floating-form #.


#DF implementation-negated(val)

    "{($val$)is-string-of-ones-and-zeroes}"

    => ($($val$)complemented$) incremented-by-one #.


#DF implementation-floating-product(val-x,"+"val-y)
```

```
     "{($\val-x,val-y\$)
     are-implementation-numeric-representations}"

     => ($normalized-floating-product-process
        (($extended-precision-form-of(val-x)$)
        with-mantissa-right-shifted,"*"
        ($extended-precision-form-of(val-y)$)
        with-mantissa-right-shifted)$)
        converted-to-implementation-floating-form #.


#DF normalized-floating-product-process
        (impl-val-x,"*"impl-val-y)

     "{($ \impl-val-x, impl-val-y\ $)
     are-extended-precision-forms}"

     => ($ \implementation-integer-add
        (implementation-integer-add
        (floating-exponent-from(impl-val-x), "+"
        floating-exponent-from(impl-val-y) ), "+"
        ($bits-in-floating-exponent + 1$)
        converted-to-implementation-form),
        implementation-double-word-product
        (floating-mantissa-from(impl-val-x), "*"
        floating-mantissa-from (impl-val-y) ) \ $)
        normalized #.


"The numeric quantity 'bits-in-floating-exponent + 1' has
been added to the exponent to correctly adjust the binary
point for the operand representation
'with-mantissa-right-shifted'."


#DF with-mantissa-right-shifted(impl-val)

     "{($impl-val$)is-extended-precision-form}"

     =>  \ floating-exponent-from(impl-val),
        implementation-right-arithmetic-shift
        (floating-mantissa-from(impl-val),"by"
        bits-per-word)\ #.


#DF implementation-double-word-product(dpval-x,"and"dpval-y)

     "{($\dpval-x,dpval-y\$)
```

```
     are-implementation-double-word-numeric-reps}"

     => ($($dpval-x$)converted-to-standard-form * ($
        dpval-y$) converted-to-standard-form $)
        with-result-converted-to-implementation-dp-form #.


#DF implementation-floating-quotient(val-x,"/"val-y)

     "{($\val-x,val-y\$)
     are-implementation-numeric-representations}"

     => ($normalized-floating-quotient-process
        (extended-precision-form-of(val-x),"/"
        ($extended-precision-form-of(val-y)$)
        with-mantissa-right-shifted)$)
        converted-to-implementation-floating-form #.


#DF normalized-floating-quotient-process
        (impl-val-x, "/" impl-val-y)

     "{ ($ \impl-val-x, impl-val-y \ $)
     are-extended-precision-forms}"

     => ($ \implementation-integer-add
        (implementation-integer-subtract
        (floating-exponent-from(impl-val-x), "-"
        floating-exponent-from(impl-val-y)), "+" ($
        bits-in-floating-mantissa - 1 $)
        converted-to-implementation-form),
        implementation-double-word-quotient
        (floating-mantissa-from(impl-val-x), "/"
        floating-mantissa-from(impl-val-y) ) \ $) normalized
        #.
```

"The numeric quantity 'bits-in-floating-mantissa - 1' has been added to the exponent to correctly adjust the binary point for the divisor representation 'with-mantissa-right-shifted'."

```
#DF implementation-double-word-quotient
        (dpval-x,"/"dpval-y)

     "{($\dpval-x,dpval-y\$)
     are-implementation-double-word-numeric-reps}"
```

```
     => ($($dpval-x$)converted-to-standard-form /
        ($dpval-y$)converted-to-standard-form$)
        with-result-converted-to-implementation-dp-form #.


#DF are-extended-precision-forms(seq)

     "{seq #IS #SEQUENCE #AND #LENGTH(seq)>=1}"

     => ($#FIRST-ELEMENT-IN seq $)
        is-extended-precision-form #IF #LENGTH(seq)=1;

     => ($#FIRST-ELEMENT-IN seq $)
        is-extended-precision-form #AND
        ($all-but-first-element-in(seq)$)
        are-extended-precision-forms #OTHERWISE #.


#DF is-extended-precision-form(impl-val)

     => ($floating-exponent-from(impl-val)$)
        is-implementation-numeric-representation #AND
        ($floating-mantissa-from(impl-val)$)
        is-implementation-double-word-numeric-representation
        #AND #LENGTH(impl-val)=2 #.


#DF implementation-special-exponential(val-x,"**"val-y)

     "{($val-x,val-y$)
     are-implementation-floatingrepresentations}"

     => #EXTERNAL-CALL-OF
        'external-implementation-special-exponential'
        #WITH-ARGUMENT(\val-x,"**"val-y\) #.


#DF implementation-floating-exponential(val-x,"**"val-y)

     "{($val-x,val-y$)
     are-implementation-floating-representations}"

     =>
        #EXTERNAL-CALL-OF'external-implementation-floating-exponential'
        #WITH-ARGUMENT(\val-x,"**"val-y\) #.
```

```
#DF integer-portion-indicated-in-floating-constant (unit)

     "{unit #IS <floating-constant> #AND unit #EQ
     current-executable-unit}"

     => simple-integer-portion-from
        (floating-constant-form-of(unit)) #IF
        floating-constant-form-of(unit) #IS
        <simple-floating-constant>;

     => exponentiated-integer-portion-from
        (floating-constant-form-of(unit)) #IF
        floating-constant-form-of(unit) #IS
        <exponentiated-floating-constant> #.


#DF fraction-portion-indicated-in-floating-constant(unit)

     "{unit #IS <floating-constant> #AND unit #EQ
     current-executable-unit}"

     => simple-fractional-portion-from
        (floating-constant-form-of(unit)) #IF
        floating-constant-form-of(unit) #IS
        <simple-floating-constant>;

     => exponentiated-fractional-portion-from
        (floating-constant-form-of(unit)) #IF
        floating-constant-form-of(unit) #IS
        <exponentiated-floating-constant> #.

#DF binary-exponent-for(unit)

     "{unit #IS <floating-constant> #AND unit #EQ
     current-executable-unit}"

     => ($ base-2-exponent-for(unit)$)
        converted-to-implementation-form #.


#DF binary-mantissa-for(unit)

     "{unit #IS <floating-constant> #AND unit #EQ
     current-executable-unit}"

     => '0' #CW
        integer-portion-of-floating-constant-in-binary
```

```
        (unit) #CW
        fraction-portion-of-floating-constant-in-binary
        (unit) #.


#DF floating-constant-form-of(unit)

     "{ unit #IS <floating-constant>}"

     => #SEG 1 #OF unit #.


#DF base-2-exponent-for(unit)

     "{unit #IS <floating-constant> #AND unit #EQ
     current-executable-unit}"

     => #LENGTH
        (integer-portion-of-floating-constant-in-binary
        (unit) ) #IF
        integer-portion-indicated-in-floating-constant(unit)
        #N = 0;

     => #NEG (#LENGTH(#PREFIX-OF-FIRST '1' #IN
        (fraction-portion-of-floating-constant-in-binary
        (unit) ))) #IF
        fraction-portion-indicated-in-floating-constant
        (unit) #N=0 #.


#DF integer-portion-of-floating-constant-in-binary
        (unit)

     "{unit #IS <floating-constant> #AND unit #EQ
     current-executable-unit}"

     =>
        ($integer-portion-indicated-in-floating-constant(unit)$)
        converted-to-binary-form #.


"a zero integer portion yields a #NIL string."


#DF fraction-portion-of-floating-constant-in-binary
        (unit)

     "{unit #IS <floating-constant> #AND unit #EQ
```

```
    current-executable-unit}"

    => fractional-binary-representation-of
       (fraction-portion-indicated-in-floating-constant(unit),"to"
       bits-in-floating-mantissa - 1 -
       base-2-exponent-for(unit) "bits-of-precision") #IF
       integer-portion-indicated-in-floating-constant(unit)
       #N=0;

    => binary-fraction-of
       (fraction-portion-indicated-in-floating-constant(unit),"to"
       bits-in-floating-mantissa - 1 "significant-bits")
       #IF
       fraction-portion-indicated-in-floating-constant(unit)
       #N=0 #.


"a non-zero integer-portion and a zero fractional portion
will give a binary representation of the fraction portion of
the correct number of zeroes to complete the mantissa."


#DF converted-to-binary-form(sem-const)

    "{($sem-const$) is-semanol-decimal-integer-constant
    #AND sem-const >= 0}"

    => #NIL #IF sem-const=0;

    => #PREFIX-OF-FIRST '#B2' #IN #CONVERT 2(sem-const) #IF
       sem-const>0 #.


#DF binary-fraction-of(d,"to" e "significant-bits")

    "{d #IS-IN #NAT-NOS #AND e>0}"

    => next-binary-digit-from (d) #CW binary-fraction-of
       (binary-fractional-residue (d), "to" e
       "significant-bits") #IF next-binary-digit-from(d)=0;

    => next-binary-digit-from(d) #CW
       fractional-binary-representation-of
       (binary-fractional-residue(d),"to" e - 1
       "bits-of-precision") #OTHERWISE #.
```

```
#DF are-implementation-numeric-representations (seq)

     "{ seq #IS #SEQUENCE #AND #LENGTH (seq) >=1}"

     => ($ #FIRST-ELEMENT-IN seq$)
        is-implementation-numeric-representation #IF #LENGTH
        (seq) = 1;

     => ($ #FIRST-ELEMENT-IN seq $)
        is-implementation-numeric-representation #AND ($
        all-but-first-element-in (seq)$)
        are-implementation-numeric-representations
        #OTHERWISE #.


#DF is-implementation-numeric-representation (x)

     => #TRUE #IFF ($x$) is-string-of-ones-and-zeroes #AND
        #LENGTH (x) = bits-per-word #.


#DF are-strings-of-ones-and-zeroes (seq)

     "{seq #IS #SEQUENCE}"

     => #TRUE #IF seq #EQ #NILSEQ;

     => ($ #FIRST-ELEMENT-IN seq $)
        is-string-of-ones-and-zeroes #AND
        ($all-but-first-element-in (seq)$)
        are-strings-of-ones-and-zeroes #OTHERWISE #.


#DF is-string-of-ones-and-zeroes (x)

     => #FALSE #IF x #IS-NOT #STRING;

     => #TRUE #IFF x #IS <binary-string> #OTHERWISE #.


#DF is-implementation-double-word-numeric-representation (x)

     => #TRUE #IFF ($ x $) is-string-of-ones-and-zeroes #AND
        #LENGTH (x) = 2 * bits-per-word #.
```

```
#DF are-attributes (seq)

     "{ seq #IS #SEQUENCE #AND #LENGTH (seq) >= 1}"

     => ($ #FIRST-ELEMENT-IN seq $) is-attribute #IF #LENGTH
        (seq) =1;

     => ($ #FIRST-ELEMENT-IN seq$) is-attribute #AND ($
        all-but-first-element-in (seq)$) are-attributes
        #OTHERWISE #.


#DF is-attribute (x)

     => #FALSE #IF x #IS-NOT #SEQUENCE;

     => #FALSE #IF #LENGTH (x) #N= 3;

     => #TRUE #IF #FOR-ALL element #IN x #IT-IS-TRUE-THAT
        (element #IS #INTEGER) ;

     => #FALSE #OTHERWISE #.


#DF is-semanol-base-2-integer-constant (x)

     => #FALSE #IF x #IS-NOT #INTEGER;

     => #TRUE #IFF ( #RIGHT 3 #CHARACTERS-OF x) #EQW
        '#B2'#OTHERWISE #.


#DF are-semanol-base-2-integer-constants(seq)

     "{ seq #IS #SEQUENCE #AND #LENGTH(seq) >= 1 }"

     =>($ #FIRST-ELEMENT-IN seq $)
        is-semanol-base-2-integer-constant #IF #LENGTH(seq)
        = 1 ;

     => ($ #FIRST-ELEMENT-IN seq $)
        is-semanol-base-2-integer-constant #AND ($
        all-but-first-element-in (seq) $)
        are-semanol-base-2-integer-constants #OTHERWISE #.
```

```
" #DF is-a-program-context (nx) => #TRUE #IFF ($ nx $)
is-an-actual-program-context #OR nx #IS
<selected-compool-name> #. "

#DF is-an-actual-program-context (nx)

    => #TRUE #IFF ($ nx $) is-a-program-unit #.

#DF is-a-program-unit (nx)

    "{ nx #IS #NODE }"

    => #TRUE #IFF nx #IS <program> #U
       <procedure-declaration> #U <procedure-subprogram> #U
       <close-subprogram> #U <implementation-compool> #U
       <defaults> #.

#DF actual (context)

    "{ ($ context $) is-a-program-context}"

    => #FIRST cp #IN sequence-of-compools-in
       (system-containing (context)) #SUCH-THAT (name-of
       (cp) #EQW context) #IF context #IS
       <selected-compool-name>;

    => context #IF ($ context $)
       is-an-actual-program-context #.

#DF sequence-of-compools-in (sys)

    "{sys #IS <jovial-j3-system>}"

    => #SEQUENCE-OF <implementation-compool> #IN sys #.

#DF program-context (nx)

    "{ nx #IS #NODE}"

    => next-outer-context (nx) #IF ($ nx $)
       is-an-outermost-program-unit #OR nx #IS
```

```
        <selected-compool-name>;

     => program-unit-containing (nx) #OTHERWISE #.


#DF is-an-outermost-program-unit (px)

     "{px #IS #NODE}"

     => #TRUE #IFF ($px$) is-a-program-unit #AND px #IS-NOT
        <procedure-declaration> #.


#DF next-outer-context (nx)

     "{ ($ nx $) is-an-outermost-program-unit #OR nx #IS
     <selected-compool-name>}"

     => next (nx, "in" sequence-of-outer-contexts-for (nx))
        #IF nx #IS <selected-compool-name>;

     => #FIRST-ELEMENT-IN sequence-of-outer-contexts-for
        (nx) #IF nx #IS <program> #U <close-subprogram> #U
        <procedure-subprogram> #.


#DF sequence-of-outer-contexts-for (nx)

     => #SEQUENCE-OF <selected-compool-name> #IN
        control-input-of (program-unit-containing (nx)) #CS
        \ defaults-of (system-containing (nx)) \ #.


#DF defaults-of (sys)

     "{ sys #IS <jovial-j3-system>}"

     => #SEG 6 #OF sys #.


#DF next (nx, "in" seq)

     "{nx #IS #NODE #AND nx #IS-IN seq}"

     =>((#ORDPOSIT nx #IN seq)+1) #TH-ELEMENT-IN seq #.


#DF program-unit-containing (nx)
```

```
    "{nx #IS #NODE}"

    => #LAST px #IN ( #SEQUENCE-OF-ANCESTORS-OF (nx)  )
       #SUCH-THAT (($px$)is-a-program-unit) #.

#DF control-input-of (progunit)

    "{progunit #IS <program> #U <close-subprogram> #U
    <procedure-subprogram>}"

    => #SEG 1 #OF #PARENT-NODE (progunit) #.
```

```
#DF is-declared ( nx,"in"decl)

    "{ #STRING-OF-TERMINALS-OF nx #IS<name> #AND
    ($decl$)is-a-declaration-for-a-name}"

    => is-declared-in-category-1-decl(nx,"in"decl)
       #IF($decl$)is-category-1-declaration;

    => is-declared-in-category-2-decl(nx,"in"decl) #IF
       ($decl$)is-category-2-declaration;

    => is-declared-in-category-3-decl(nx,"in" decl) #IF
       ($decl$)is-category-3-declaration #.

#DF is-category-1-declaration(decl)

    => #FALSE#.

#DF is-declared-in-category-1-decl(nx,"in"decl)

    => #FALSE#.

#DF is-category-2-declaration(decl)

    "{decl #IS #NODE}"

    => #TRUE #IFF decl #IS-IN
       sequence-of-outer-category-2-decls-in ( actual
       (program-context(decl)) ) #.

#DF is-declared-in-category-2-decl (nx,"in" decl )

    "{ #STRING-OF-TERMINALS-OF nx #IS<name> #AND ($ decl
    $)is-category-2-declaration}"

    => #THERE-EXISTS stmt-name #IN #SEQUENCE-OF
       <statement-name> #IN
       optional-statement-name-list-of( decl ) #SUCH-THAT
       (#STRING-OF-TERMINALS-OF(nx) #EQW
       #STRING-OF-TERMINALS-OF(stmt-name)) #IF decl #IS
       <statement>;
```

```
    => #STRING-OF-TERMINALS-OF(nx) #EQW
       #STRING-OF-TERMINALS-OF(name-declared-by( decl ))
       #OTHERWISE #.


#DF optional-statement-name-list-of(stmt)

    "{stmt #IS<statement>}"

    => #SEG 1 #OF stmt #.


#DF name-declared-by (decl)

    "{($decl$) is-category-2-declaration}"

    => decl #IF decl #IS <formal-input-close-parameter> #U
       <formal-output-destination-parameter>;

    => #SEG 3 #OF decl #IF decl #IS <close-declaration> #U
       <program-declaration> #U <item-switch-declaration>
       #U <index-switch-declaration> #.


#DF a-category-2-declaration-exists-for (dn)

    "{ dn #IS <destination-name> #U
    <actual-input-close-parameter> #U
    <actual-output-destination-parameter> }"

    => there-exists-c2-decl-for (dn, "looking-first-in"
       program-context (dn)) #.


#DF there-exists-c2-decl-for (dname, "looking-first-in"
context)

    "{ dname #IS <destination-name> #U
    <actual-input-close-parameter> #U
    <actual-output-destination-parameter> #AND
    ($context$)is-a-program-context}"

    => #TRUE #IF #THERE-EXISTS candidate #IN
       sequence-of-outer-category-2-decls-in (actual
       (context)) #SUCH-THAT(($dname, "in" candidate$)
       is-declared);

    => there-exists-c2-decl-for (dname, "looking-next-in"
```

```
    program-context(context) ) #IF
    ($context$)is-not-the-outermost-context;

  => #FALSE #OTHERWISE#.


#DF is-not-the-outermost-context (context)

  "{ ($ context $) is-a-program-context}"

  => #TRUE #IFF context #IS-NOT <implementation-compool>
     #U <defaults> #.


#DF category-2-declaration-for (dname)

  "{dname #IS <destination-name> #U
  <actual-input-close-parameter> #U
  <actual-output-destination-parameter>}"

  => c2-declaration-for (dname,"in" program-context
     (dname)) #.


#DF c2-declaration-for(dname,context)

  "{ dname #IS <destination-name> #U
  <actual-input-close-parameter> #U
  <actual-output-destination-parameter> #AND ($context$)
  is-a-program-context}"

  => #LAST candidate #IN
     sequence-of-outer-category-2-decls-in (actual
     (context)) #SUCH-THAT
     (($dname,"in"candidate$)is-declared) #IF
     #THERE-EXISTS candidate #IN
     sequence-of-outer-category-2-decls-in (actual
     (context))
     #SUCH-THAT(($dname,"in"candidate$)is-declared);

  => c2-declaration-for (dname, program-context(context))
     #OTHERWISE#.


#DF sequence-of-outer-category-2-decls-in (px)

  "{($px$)is-a-program-context}"
```

```
    => sequence-of-category-2-decls-in(px) #IF px #IS-NOT
       <jovial-j3-program>;

    => #SUBSEQUENCE-OF-ELEMENTS decl #IN
       sequence-of-category-2-decls-in (px) #SUCH-THAT
       (($decl$) is-not-in-a-procedure-declaration)
       #OTHERWISE #.


#DF sequence-of-category-2-decls-in(px)

    "{($px$)is-a-program-context}"

    => #SEQUENCE-OF <statement> #U <close-declaration> #U
       <item-switch-declaration> #U
       <index-switch-declaration> #U <program-declaration>
       #U <formal-input-close-parameter> #U
       <formal-output-destination-parameter> #IN px #.


#DF is-not-in-a-procedure-declaration (nx)

    "{nx #IS #NODE}"

    => #TRUE #IFF #FOR-ALL ancestor #IN
       #SEQUENCE-OF-ANCESTORS-OF (nx) #IT-IS-TRUE-THAT
       (ancestor #IS-NOT <procedure-declaration>) #.


#DF is-category-3-declaration(decl)

    => #TRUE #IFF ($ decl $)
       is-explicit-category-3-declaration #OR decl #IS
       <mode-directive> #.


#DF is-declared-in-category-3-decl(nx,"in"decl)

    => #TRUE #IFF ($ nx, "in" decl $) is-properly-declared
       #.


#DF a-category-3-declaration-exists-for (var)

    "{ var #IS <simple-variable> #U <indexed-variable> #U
    <name>}"

    => #TRUE #IFF ($ variable-name-of (var) $)
```

```
        has-an-explicit-c3-declaration #OR ($
        variable-name-of (var) $)
        has-a-declaring-mode-directive
         #.


#DF has-an-explicit-c3-declaration (var)

      "{ var #IS <name>}"

      => #TRUE #IFF there-exists-explicit-c3-declaration-for
         (var, "looking-first-in" program-context (var)) #.


#DF has-a-declaring-mode-directive (var)

      "{ var #IS <name>}"

      => #TRUE #IFF there-exists-mode-directive-declaring
         (var, "looking-first-in" program-context (var)) #.


#DF there-exists-mode-directive-declaring
         (var, "looking-first-in" context)

      "{ var #IS <name> #AND ($ context $)
      is-a-program-context}"

      => #TRUE #IF #THERE-EXISTS candidate #IN
         sequence-of-mode-directives-in (actual (context) )
         #SUCH-THAT ( ($ var, "in" candidate $) is-declared);

      => there-exists-mode-directive-declaring (var, "looking
         next in" program-context (context)) #IF ($ context
         $) is-not-the-outermost-context;

      => #FALSE #OTHERWISE #.


#DF category-3-declaration-for (var)

      "{ var #IS <simple-variable> #U <indexed-variable> #U
      <name> }"

      => explicit-c3-declaration-for (variable-name-of (var)
         , "looking-first-in" program-context (var)) #IF ($
         variable-name-of (var) $)
         has-an-explicit-c3-declaration ;
```

```
     => mode-directive-declaring (variable-name-of (var),
        "looking-first-in" program-context (var)) #OTHERWISE
        #.

#DF variable-name-of (nx)

     "{ nx #IS <simple-variable> #U <indexed-variable> #U
     <name> }"

     => nx #IF nx #IS <name> ;

     => #SEG 1 #OF nx #OTHERWISE #.

#DF explicit-c3-declaration-for (nx, "looking-first-in"
context)

     "{nx #IS <name> #AND ($context$) is-a-program-context}"

     => #LAST candidate #IN
        sequence-of-explicit-outer-category-3-decls-in
        (actual (context)) #SUCH-THAT (($nx, "in"
        candidate$) is-declared) #IF #THERE-EXISTS candidate
        #IN sequence-of-explicit-outer-category-3-decls-in
        (actual (context)) #SUCH-THAT (($nx, "in"
        candidate$) is-declared);

     => explicit-c3-declaration-for (nx, "looking-next-in"
        program-context (context)) #OTHERWISE #.

#DF there-exists-explicit-c3-declaration-for (nx,
"looking-in" context)

     "{nx #IS <name> #AND ($context$) is-program-context}"

     => #TRUE #IF #THERE-EXISTS candidate #IN
        sequence-of-explicit-outer-category-3-decls-in
        (actual (context)) #SUCH-THAT (($nx, "in"
        candidate$) is-declared);

     => there-exists-explicit-c3-declaration-for (nx,
        "looking-next-in" program-context (context) ) #IF
        ($context$) is-not-the-outermost-context;

     => #FALSE #OTHERWISE #.
```

```
#DF sequence-of-explicit-outer-category-3-decls-in (context)

    "{($context$) is-an-actual-program-context}"

    => sequence-of-explicit-category-3-decls-in (context)
       #IF context #IS-NOT <jovial-j3-program>;

    => #SUBSEQUENCE-OF-ELEMENTS decl #IN
       sequence-of-explicit-category-3-decls-in (context)
       #SUCH-THAT (($decl$)
       is-not-in-a-procedure-declaration) #OTHERWISE #.


#DF sequence-of-explicit-category-3-decls-in (context)

    "{($context$) is-an-actual-program-context}"

    => #SEQUENCE-OF-NODES nx #IN context #SUCH-THAT (($nx$)
       is-explicit-category-3-declaration) #.


#DF is-explicit-category-3-declaration (nx)

    "{nx #IS #NODE}"

    => #TRUE #IFF ($nx$) is-data-declaration #OR nx #IS
       <file-declaration> #U <procedure-declaration>  #U
       <subprogram-declaration> #.

#DF is-data-declaration (nx)

    "{nx #IS #NODE}"

    => #TRUE #IFF nx #IS <simple-item-declaration> #U
       <array-declaration> #U <ordinary-table-declaration>
       #U <defined-entry-table-declaration> #U
       <like-table-declaration> #U
       <ordinary-table-item-declaration> #U
       <string-item-declaration> #U
       <defined-entry-item-declaration> #.

#DF mode-directive-declaring (nx, "looking-first-in"
context)
```

```
    "{nx #IS <name> #AND ($context$) is-a-program-context}"

    => #LAST candidate #IN sequence-of-mode-directives-in
       (actual (context)) #SUCH-THAT (($nx, "in"
       candidate$) is-declared) #IF #THERE-EXISTS candidate
       #IN sequence-of-mode-directives-in (actual
       (context)) #SUCH-THAT (($nx, "in" candidate$)
       is-declared);

    => mode-directive-declaring (nx, "looking-next-in"
       program-context (context)) #OTHERWISE #.


#DF sequence-of-mode-directives-in (context)

    "{($context$) is-an-actual-program-context}"

    => #SEQUENCE-OF <mode-directive> #IN context #.


#DF is-properly-declared (nx, "in" decl)

    "{nx #IS <name> #AND ($decl$)
    is-explicit-category-3-declaration}"

    => #FALSE #IF nx #PRECEDES decl #IN actual
       (program-context (decl)) #AND ($ nx $)
       is-not-formal-parameter-name ;

    => #TRUE #IF #STRING-OF-TERMINALS-OF (nx) #EQW
       #STRING-OF-TERMINALS-OF (name-declared-in (decl) ) ;

    => ($#STRING-OF-TERMINALS-OF (nx), "by" decl$)
       is-declared-implicitly #IF decl #IS
       <like-table-declaration>;

    => #FALSE #OTHERWISE #.


#DF is-not-formal-parameter-name (nx)

    "{ nx #IS <name> }"

    => #PARENT-NODE (nx) #IS-NOT <formal-input-parameter>
       #U <formal-output-parameter> #.


#DF name-declared-in (decl)
```

```
    "{($decl$) is-explicit-category-3-declaration}"

    => #SEG 3 #OF decl #IF decl #IS <file-declaration> #U
       <subprogram-declaration> #U
       <simple-item-declaration> #U <array-declaration> #U
       <ordinary-table-declaration> #U
       <defined-entry-table-declaration> #U
       <like-table-declaration> #U
       <ordinary-table-item-declaration> #U
       <string-item-declaration> #U
       <defined-entry-item-declaration>;

    => #SEG 3 #OF (procedure-head-of (decl)) #IF decl #IS
       <procedure-declaration> #.


#DF is-declared-implicitly (nx, "by" decl)

    "{decl #IS <like-table-declaration> #AND nx #IS
    <name>}"

    => #FALSE #IF #LAST-CHARACTER-IN (name-declared-in
       (decl)) #NEQW #LAST-CHARACTER-IN (nx);

    => is-declared-implicitly (all-but-last-character-in
       (nx), "in" pattern-table-decl-for (decl)) #IF
       pattern-table-decl-for (decl) #IS
       <like-table-declaration>;

    => #TRUE #IFF #THERE-EXISTS item-decl #IN
       sequence-of-items-in-table (pattern-table-decl-for
       (decl)) #SUCH-THAT (name-declared-in (item-decl)
       #EQW all-but-last-character-in (nx)) #OTHERWISE #.


#DF all-but-last-character-in (str)

    "{str #IS #STRING}"

    => #NIL #IF str #IS #NIL;

    => #LEFT (#LENGTH (str) - 1) #CHARACTERS-OF str
       #OTHERWISE #.


#DF pattern-table-decl-for (decl)
```

```
     "{decl #IS <like-table-declaration>}"

     => pattern-table-for (decl,"looking-first-in"
        program-context (decl)) #.


#DF pattern-table-for (decl, "looking-first-in" context)

     "{decl #IS <like-table-declaration> #AND ($context$)
     is-program-context}"

     => #LAST candidate #IN seq-of-table-decls-in (actual
        (context)) #SUCH-THAT (($candidate, "for" decl$)
        is-a-pattern-table) #IF #THERE-EXISTS candidate #IN
        seq-of-table-decls-in (actual (context)) #SUCH-THAT
        (($candidate, "for" decl$) is-a-pattern-table);

     => pattern-table-for (decl, "looking-next-in"
        program-context (context) ) #OTHERWISE #.


#DF seq-of-table-decls-in (context)

     "{($context$) is-an-actual-program-context}"

     => #SEQUENCE-OF-NODES x #IN context #SUCH-THAT (($x$)
        is-a-table-declaration) #.


#DF is-a-table-declaration (nx)

     "{nx #IS #NODE}"

     => #TRUE #IFF nx #IS <ordinary-table-declaration> #U
        <defined-entry-table-declaration> #U
        <like-table-declaration> #.


#DF is-a-pattern-table (table-decl, "for" like-decl)

     "{like-decl #IS <like-declaration> #AND ($table-decl$)
     is-a-table-declaration}"

     => #FALSE #IF like-decl #PRECEDES table-decl #IN actual
        (program-context (table-decl)) ;

     => #TRUE #IFF all-but-last-character-in
        (name-declared-in (like-decl)) #EQW name-declared-in
```

```
        (table-decl) #OTHERWISE #.


#DF sequence-of-items-in-table (decl)

     "{decl #IS <ordinary-table-declaration> #U
     <defined-entry-table-declaration>}"

     => #SEQUENCE-OF <ordinary-table-item-declaration> #U
        <string-item-declaration> #U
        <defined-entry-item-declaration> #IN decl #.


#DF declaration-for (nx)

     "{ nx #IS <simple-variable> #U <indexed-variable> #U
     <name> }"

     => category-3-declaration-for (nx) #.


#DF detailed-declaration-for(nx)

     "{nx #IS <simple-variable> #U <indexed-variable>}"

     => counterpart-declaration-for (nx) #IF ($nx$)
        is-like-declared;

     => declaration-for (nx) #OTHERWISE #.


#DF is-like-declared (nx)

     "{ nx #IS <simple-variable> #U <indexed-variable>}"

     => #TRUE #IFF declaration-for(nx) #IS
        <like-table-declaration> #.


#DF counterpart-declaration-for(nx)

     "{ ($nx$) is-like-declared #AND nx #IS
     <simple-variable> #U <indexed-variable>}"

     => counterpart-table-declaration-for (nx) #IF
        variable-name-of (nx)#EQW
        name-declared-in(declaration-for(nx));
```

```
    => counterpart-table-item-declaration-for(nx)
       #OTHERWISE #.


#DF counterpart-table-declaration-for(nx)

    "{nx #IS <simple-variable> #U <indexed-variable>}"

    => ultimate-pattern-table-decl-for(declaration-for(nx))
       #.


#DF ultimate-pattern-table-decl-for(decl)

    "{decl #IS <like-table-declaration>}"

    => pattern-table-decl-for (decl) #IF
       pattern-table-decl-for(decl) #IS-NOT
       <like-table-declaration>;

    => ultimate-pattern-table-decl-for
       (pattern-table-decl-for(decl)) #OTHERWISE #.


#DF counterpart-table-item-declaration-for(nx)

    "{nx#IS<simple-variable> .u<indexed-variable>}"

    =>ultimate-item-decl-for (#STRING-OF-TERMINALS-OF
       (variable-name-of(nx)),"looking-first-in"
       declaration-for(nx)) #.


#DF ultimate-item-decl-for (nx, "starting-at" decl)

    "{($decl$) is-a-table-declaration #AND nx #IS #STRING}"

    => ultimate-item-decl-for (all-but-last-character-in
       (nx),

    "trying-next" pattern-table-decl-for (decl))
       #IF decl #IS <like-table-declaration>;

    => #FIRST item-decl #IN sequence-of-items-in-table
       (decl) #SUCH-THAT(name-declared-in (item-decl) #EQW
       all-but-last-character-in (nx)) #OTHERWISE #.
```

```
#DF generalized-assign-latest-value (s-ref-addr, val)

      => distributed-memory-assign (val, "to" s-ref-addr) #.


#DF distributed-memory-assign (memval, "to" s-ref-addr)

      => segmented-assignment-of (memval, "using"
         field-descriptor-sequence-implied-in (s-ref-addr))
         #.


#DF field-descriptor-sequence-implied-in (s-ref-addr)

      => descriptor-seq-for (nr-of-bits-in (s-ref-addr),
         "bits-of" s-ref-addr) #.


#DF descriptor-seq-for (bit-ct, "bits-of" s-ref-addr)

      => \ field-descriptor-of (s-ref-addr) \ #IF
         nr-of-bits-in-field-from (s-ref-addr) = bit-ct;

      => \ ($field-descriptor-of (s-ref-addr) , "with"
         bit-ct$) replacing-fld-length \ #IF
         nr-of-bits-in-field-from (s-ref-addr) > bit-ct;

      => \ field-descriptor-of (s-ref-addr) \ #CS
         descriptor-seq-for (bit-ct -
         nr-of-bits-in-field-from (s-ref-addr) , "bits-of"
         ($s-ref-addr$) adjusted-to-next-field) #OTHERWISE #.


#DF adjusted-to-next-field (s-ref-addr)

      => ($ ($ s-ref-addr, "with"
         successor-field-descriptor-of (s-ref-addr) $)
         replacing-field-descriptor, "and" (nr-of-bits-in
         (s-ref-addr) - nr-of-bits-in-field-from (s-ref-addr)
         ) $) replacing-nr-of-bits #.


#DF successor-field-descriptor-of (s-ref-addr)

      => build-field-descriptor (word-address-from
         (s-ref-addr) , next-field-first-bit-from
```

```
        (s-ref-addr) , nr-of-bits-in-next-field-from
        (s-ref-addr) ) #IF next-field-first-bit-from
        (s-ref-addr) + nr-of-bits-in-next-field-from
        (s-ref-addr) <= bits-per-word;

     => build-field-descriptor (next-word-address-from
        (s-ref-addr) , first-field-first-bit-from
        (s-ref-addr) , nr-of-bits-in-next-field-from
        (s-ref-addr) ) #OTHERWISE #.


#DF next-field-first-bit-from (s-ref-addr)

     => field-first-bit-from (s-ref-addr) +
        bits-between-field-first-bits-from (s-ref-addr) #.


#DF next-word-address-from (s-ref-addr)

     => word-address-from (s-ref-addr) +
        words-between-fields-from (s-ref-addr) #.


#DF segmented-assignment-of (memval, "using" fdseq)

     => sequential-assignment-of (seq-of-value-chunks-from
        (memval, "using" fdseq) ,"to-flds-described-in"
        fdseq) #.


#PROC-DF sequential-assignment-of (valseq,
"to-flds-described-in"
  fdseq)

  #BEGIN

    #FOR-ALL i : 1 <= i <= #LENGTH (fdseq) #DO #COMPUTE!
    assign-to-field (i#TH-ELEMENT-IN fdseq, "the-value" i
    #TH-ELEMENT-IN valseq)

    #RETURN-WITH-VALUE! #NIL

  #END #.


#DF assign-to-field (fld-desc, "the-value" val)

     => assign-latest-memory-value (word-address-of
```

```
   (fld-desc), "the-value" splice (val, "into"
   latest-memory-value (word-address-of (fld-desc)),
   "at" field-first-bit-of (fld-desc)))#.


#DF splice (newval, "into" wordval, "at-bit" n)

 => (#LEFT n #CHARACTERS-OF wordval) #CW newval #CW (
    #RIGHT (bits-per-word - (#LENGTH (newval)+n))
    #CHARACTERS-OF wordval ) #.


#DF seq-of-value-chunks-from (memval, "using" fdseq)

 => #NILSEQ #IF memval #EQW #NIL #OR fdseq #EQS #NILSEQ;

 => \ #LEFT nr-of-bits-in-field-of (first-descriptor-in
    (fdseq)) #CHARACTERS-OF memval \ #CS
    seq-of-value-chunks-from (#RIGHT (#LENGTH (memval) -
    nr-of-bits-in-field-of (first-descriptor-in
    (fdseq))) #CHARACTERS-OF memval,
    all-but-first-element-in (fdseq)) #OTHERWISE#.


#DF first-descriptor-in (fdseq)

 => #FIRST-ELEMENT-IN (fdseq) #.


#DF latest-memory-value (word-addr)

 => implementation-initial-value-at
    (uniform-notation-for (word-addr)) #IF #LATEST-VALUE
    (uniform-notation-for (word-addr)) #IS #UNDEFINED;

 => #LATEST-VALUE (uniform-notation-for (word-addr))
    #OTHERWISE#.


#DF assign-latest-memory-value (word-addr, val)

 => #ASSIGN-LATEST-VALUE (uniform-notation-for
    (word-addr), "receives" val) #.


#DF implementation-initial-value-at (word-addr)

 => ($ bits-per-word$) zeroes #.
```

```
#DF uniform-notation-for ( x )

     => #CONVERT 10 ( x ) #IF x #IS #INTEGER;

     => x #OTHERWISE #.


#DF generalized-latest-value (s-ref-addr)

     "{ ($s-ref-addr$) is-standard-reference-address}"

     => collected-memory-value (s-ref-addr) #IF
        nr-of-bits-in-field-from (s-ref-addr) #N= 0;

     => '0' #OTHERWISE #.


#DF collected-memory-value (s-ref-addr)

     => value-collected-using
        (field-descriptor-sequence-implied-in (s-ref-addr))
        #.


#DF value-collected-using (fdseq)

     => #NIL #IF fdseq #EQ #NILSEQ;

     => value-of-field-described-by (first-descriptor-in
        (fdseq)) #CW value-collected-using
        (all-but-first-element-in (fdseq)) #OTHERWISE #.


#DF value-of-field-described-by (fld-desc)

     => #LEFT nr-of-bits-in-field-of (fld-desc)
        #CHARACTERS-OF #RIGHT (bits-per-word -
        field-first-bit-of (fld-desc)) #CHARACTERS-OF
        latest-memory-value (word-address-of (fld-desc)) #.


#DF bit-addr-built-from (a,b)

     "{0<=a #AND 0<=b #AND b < bits-per-word\ " => \a,b\#.
     #DF word-part (bitaddr) "{ ($bitaddr$) is-bit-address\
     " => #FIRST-ELEMENT-IN bitaddr #.  #DF bit-part
```

```
(bitaddr) "{($bitaddr$) is-bit-address}"

=> 1 #TH-ELEMENT-IN bitaddr #.
```

```
#DF build-field-descriptor (wd-addr, fld-bit, fld-size)

     => \ wd-addr, fld-bit, fld-size \ #.

#DF word-address-of (fld-desc)

     => #FIRST-ELEMENT-IN fld-desc #.

#DF field-first-bit-of (fld-desc)

     => 2 #TH-ELEMENT-IN fld-desc #.

#DF nr-of-bits-in-field-of (fld-desc)

     => 3 #TH-ELEMENT-IN fld-desc #.

#DF replacing-word-address (fld-desc, new-addr)

     => build-field-descriptor (new-addr, field-first-bit-of
        (fld-desc), nr-of-bits-in-field-of (fld-desc)) #.

#DF replacing-field-first-bit (fld-desc, new-first-bit)

     => build-field-descriptor (word-address-of (fld desc),
        new-first-bit, nr-of-bits-in-field-of (fld-desc)) #.

#DF replacing-fld-length (fld-desc,new-length)

     => build-field-descriptor (word-address-of (fld-desc),
        field-first-bit-of (fld-desc), new-length) #.

#DF create-standard-reference-address
(fld-desc,nxt-fld-desc,nr-flds)

     => \ fld-desc, nxt-fld-desc, nr-flds \ #.

#DF field-descriptor-of (s-ref-addr)
```

```
     => #FIRST-ELEMENT-IN s-ref-addr #.


#DF next-field-descriptor-of (s-ref-addr)

     => 2 #TH-ELEMENT-IN s-ref-addr #.


#DF nr-of-bits-in (s-ref-addr)

     => 3 #TH-ELEMENT-IN s-ref-addr #.


#DF replacing-field-descriptor (s-ref-addr, new-desc)

     => create-standard-reference-address (new-desc,
        next-field-descriptor-of (s-ref-addr), nr-of-bits-in
        (s-ref-addr)) #.


#DF replacing-next-field-descriptor (s-ref-addr, new-desc)

     => create-standard-reference-address
        (field-descriptor-of (s-ref-addr), new-desc,
        nr-of-bits-in (s-ref-addr)) #.


#DF replacing-nr-of-bits (s-ref-addr, new-count)

     => create-standard-reference-address
        (field-descriptor-of (s-ref-addr),
        next-field-descriptor-of (s-ref-addr), new-count) #.


#DF word-address-from (s-ref-addr)

     => word-address-of ( field-descriptor-of(s-ref-addr))
        #.


#DF field-first-bit-from (s-ref-addr)

     => field-first-bit-of ( field-descriptor-of
        (s-ref-addr)) #.


#DF nr-of-bits-in-field-from (s-ref-addr)
```

```
     => nr-of-bits-in-field-of ( field-descriptor-of
        (s-ref-addr)) #.

#DF bits-between-field-first-bits-from (s-ref-addr)

     => bits-between-field-first-bits-of
        (next-field-descriptor-of (s-ref-addr)) #.

#DF first-field-first-bit-from (s-ref-addr)

     => first-field-first-bit-of (next-field-descriptor-of
        (s-ref-addr)) #.

#DF words-between-fields-from (s-ref-addr)

     => words-between-fields-of (next-field-descriptor-of
        (s-ref-addr)) #.

#DF nr-of-bits-in-next-field-from (s-ref-addr)

     => nr-of-bits-in-next-field-of
        (next-field-descriptor-of (s-ref-addr) ) #.

#DF build-next-field-descriptor (delta-bits, first-fld-bit,
        delta-words, bits-per-fld )

     => \delta-bits, fi st-fld-bit, delta-words,
        bits-per-fld\ #.

#DF bits-between-field-first-bits-of (nxt-fld-desc)

     => #FIRST-ELEMENT-IN nxt-fld-desc #.

#DF first-field-first-bit-of (nxt-fld-desc)

     => 2 #TH-ELEMENT-IN (nxt-fld-desc) #.

#DF words-between-fields-of (nxt-fld-desc)

     => 3 #TH-ELEMENT-IN nxt-fld-desc #.
```

```
#DF nr-of-bits-in-next-field-of (nxt-fld-desc)

    => 4 #TH-ELEMENT-IN nxt-fld-desc #.

    " #DF replacing-field-distance (nxt-fld-desc,
    new-distance) => build-next-field-descriptor
    (new-distance, first-field-first-bit-of (nxt-fld-desc),
    words-between-fields-of (nxt-fld-desc),
    nr-of-bits-in-next-field-of (nxt-fld-desc) ) #.  #DF
    replacing-first-field-bit (nxt-fld-desc, new-fst-bit)
    => build-next-field-descriptor
    (bits-between-field-first-bits-of (nxt-fld-desc),
    new-fst-bit, words-between-fields-of (nxt-fld-desc),
    nr-of-bits-in-next-field-of (nxt-fld-desc) ) #.  "


#DF replacing-word-increment (nxt-fld-desc, new-wd-inc)

    => build-next-field-descriptor
       (bits-between-field-first-bits-of (nxt-fld-desc),
       first-field-first-bit-of (nxt-fld-desc), new-wd-inc,
       nr-of-bits-in-next-field-of (nxt-fld-desc) ) #.


#DF standard-reference-address-of (nx)

    "{ ($nx$) is-a-variable #OR nx #IS <name> #U
    <loop-variable>}"

    => indexed-standard-reference-address (
       variable-name-of (nx) ,"using" index-values
       (index-list-of (nx))) #IF nx #IS <indexed-variable>;

    => simple-standard-reference-address (nx) #IF nx #IS
       <simple-variable>;

    => entry-standard-reference-address (nx) #IF nx #IS
       <entry-variable>;

    => special-int-var-reference-address (nx) #IF nx #IS
       <special-integer-variable>;

    => special-fix-var-reference-address (nx) #IF nx #IS
       <special-fixed-variable>;

    => special-literal-var-reference-address (nx) #IF nx
```

```
    #IS <special-literal-variable>;

 => special-boolean-var-reference-address (nx) #IF nx
    #IS <special-boolean-variable>;

 => simple-standard-reference-address (nx) #IF nx #IS
    <name> #U <loop-variable> #.

#DF indexed-standard-reference-address (var, index-vals)

   "{var #IS <name> #AND #FOR-ALL x #IN index-vals
   #IT-IS-TRUE-THAT (x>=0)}"

 => ($ indexed-standard-relative-address (var,
    index-vals), "with" word-address-from
    (parent-table-standard-reference-address (var))$)
    added-to-first-word-address #IF ($var$)
    is-array-reference;

 => ($ indexed-standard-relative-address (var,
    index-vals), "with" word-address-from
    (parent-table-standard-reference-address (var)) +
    one-for-nent-word$) added-to-first-word-address
    #OTHERWISE #.


#DF is-call-by-name-table-item-reference (nm)

 => ($nm$) is-that-of-a-call-by-name-parameter #IF
    ($nm$) is-array-reference;

 => ($ name-declared-in (table-implicitly-referenced-by
    (nm)) $) is-that-of-a-call-by-name-parameter
    #OTHERWISE #.


#DF call-by-name-table-reference-address (var)

 => ($ basic-addr (var), "with"
    ($generalized-latest-value
    (simple-standard-reference-address (var)) $)
    converted-to-standard-form$)
    replacing-first-word-address #IF ($var$)
    is-array-reference;

 => ($ basic-addr (name-declared-in
    (table-implicitly-referenced-by (var)) ), "with"
```

```
     word-address-from (simple-standard-reference-address
     (name-declared-in (table-implicitly-referenced-by
     (var))))$) replacing-first-word-address #OTHERWISE
     #.


#DF parent-table-standard-reference-address (var)

    "{var #IS <name>}"

    => call-by-name-table-reference-address (var) #IF
       ($var$) is-call-by-name-table-item-reference;

    => simple-standard-reference-address (var) #OTHERWISE
       #.


#DF indexed-standard-relative-address (var,index-vals)

    "{ var #IS <name> #FOR-ALL x #IN index-vals
    #IT-IS-TRUE-THAT (x>=0)}"

    => indexed-ordinary-item-relative-address (var,
       index-vals) #IF ($var$) is-ordinary-item-reference;

    => indexed-defined-item-relative-address (var,
       index-vals) #IF ($var$)
       is-defined-entry-item-reference;

    => indexed-string-item-relative-address (var,
       index-vals) #IF ($var$) is-string-item-reference;

    => indexed-array-element-relative-address
       (var,index-vals) #IF ($var$) is-array-reference #.


#DF is-ordinary-item-reference (nx)

    => detailed-declaration-for (nx)
       #IS<ordinary-table-item-declaration> #.


#DF indexed-ordinary-item-relative-address (var,index-vals)

    "{ ($var$) is-ordinary-item-reference #AND var #IS
    <name>}"

    => ($ ord-item-entry-relative-addr (var), "by"
```

```
        #FIRST-ELEMENT-IN index-vals, "and"
        number-of-entries-of (table-implicitly-referenced-by
        (var))$) modified-for-parallel-table #IF ($
        table-implicitly-referenced-by(var)$)
        is-parallel-table;

     => ($ord-item-entry-relative-addr(var) , "by"
        #FIRST-ELEMENT-IN index-vals, "and"
        number-of-words-per-entry-in
        (table-implicitly-referenced-by (var))$)
        modified-for-serial-table #OTHERWISE #.


#DF is-parallel-table (dec)

     => structure-spec-of (dec) #EQW 'p' #.


#DF structure-spec-of (dec)

     => like-table-structure-spec (dec) #IF dec #IS
        <like-table-declaration>;

     => #STRING-OF-TERMINALS-OF
        (optional-structure-specification-of (dec)) #IF dec
        #IS <ordinary-table-declaration> #U
        <defined-entry-table-declaration> #.


#DF like-table-structure-spec (dec)

     "{dec #IS <like-table-declaration>}"

     => structure-spec-of (pattern-table-decl-for (dec)) #IF
        (#STRING-OF-TERMINALS-OF
        (optional-structure-specification-of (dec)) ) #EQW
        #NIL;

     => #STRING-OF-TERMINALS-OF
        (optional-structure-specification-of (dec))
        #OTHERWISE #.


#DF optional-structure-specification-of (dec)

     "{dec #IS <like-table-declaration> #U
     <defined-entry-table-declaration> #U
     <ordinary-table-declaration>}"
```

```
    => #SEG 7 #OF dec #.

#DF modified-for-parallel-table (s-ref-addr, ix, nr-entries)

    "{($ s-ref-addr$) is-entry-relative-addr #AND ix >=0
    #AND nr-entries > 0}"

    => ($($ s-ref-addr, "with" ix$)
       replacing-first-word-address, "and" nr-entries *
       words-between-fields-from (s-ref-addr)$)
       replacing-next-word-increment #.

#DF replacing-next-word-increment (s-ref-addr, n)

    => ($ s-ref-addr, "with" ($ next-field-descriptor-of
       (s-ref-addr), "with" n$) replacing-word-increment$)
       replacing-next-field-descriptor #.

#DF modified-for-serial-table (s-ref-addr, ix,
wds-per-entry)

    "{($s-ref-addr$) is-entry-relative-addr #AND ix >=0
    #AND wds-per-entry > 0}"

    => ($s-ref-addr, "with" ix * wds-per-entry$)
       replacing-first-word-address #.

#DF ord-item-entry-relative-addr (var)

    "{($var$) is-ordinary-item-reference #AND var #IS
    <name>}"

    => dense-packed-item-entry-relative-addr (var) #IF
       ($table-implicitly-referenced-by (var)$)
       is-densely-packed;

    => med-packed-item-entry-relative-addr (var) #IF
       ($table-implicitly-referenced-by (var)$)
       is-med-packed;

    => unpacked-item-entry-relative-addr (var) #IF
       ($table-implicitly-referenced-by (var)$) is-unpacked
       #.
```

```
#DF is-unpacked (dec)

     => packing-spec-of (dec) #EQW 'N' #.

#DF packing-spec-of (dec)

     => like-table-packing-spec (dec) #IF dec #IS
        <like-table-declaration>;

     => ordinary-table-packing-spec (dec) #IF dec #IS
        <ordinary-table-declaration>;

     => defined-entry-table-packing-spec (dec) #IF dec #IS
        <defined-entry-table-declaration> #.

#DF ordinary-table-packing-spec (dec)

     => 'N' #IF ( #STRING-OF-TERMINALS-OF
        (optional-packing-specification-of (dec)) ) #EQW
        #NIL;

     => #STRING-OF-TERMINALS-OF
        (optional-packing-specification-of (dec)) #OTHERWISE
        #.

#DF defined-entry-table-packing-spec (dec)

     => 'D' #IF ( #STRING-OF-TERMINALS-OF
        (optional-packing-specification-of (dec)) ) #EQW
        #NIL;

     => #STRING-OF-TERMINALS-OF
        (optional-packing-specification-of (dec)) #OTHERWISE
        #.

#DF like-table-packing-spec (dec)

     => packing-spec-of (pattern-table-decl-for (dec) ) #IF
        ( #STRING-OF-TERMINALS-OF
        (optional-packing-specification-of (dec) ) ) #EQW
        #NIL;
```

```
    => #STRING-OF-TERMINALS-OF
       (optional-packing-specification-of (dec)) #OTHERWISE
       #.


#DF unpacked-item-entry-relative-addr (var)

    "{ ($var$) is-ordinary-item-reference }"

    => entry-rel-addr-of-unpacked-allocated-name
       (name-declared-in (detailed-declaration-for (var) )
       ) #IF ($name-declared-in (detailed-declaration-for
       (var) ) $) is-actually-allocated-in-entry;

    => overlay-element-rel-addr (first-overlay-mention-of
       (name-declared-in (detailed-declaration-for (var) )
       ) ) #IF ($name-declared-in (detailed-declaration-for
       (var) ) $) is-mentioned-in-a-subordinate-ovly-dec #.


#DF first-overlay-mention-of (nm)

    "{ ($nm$) is-mentioned-in-a-subordinate-ovly-dec }"

    => first-mention-of (nm, "in" #FIRST-ELEMENT-IN
       seq-of-subord-overlay-declarations-mentioning (nm) )
       #.


#DF is-med-packed (dec)

    => packing-spec-of (dec) #EQW 'M' #.


#DF med-packed-item-entry-relative-addr (var)

    "{ ($var$) is-ordinary-item-reference }"

    => entry-rel-addr-of-med-packed-allocated-name
       (name-declared-in (detailed-declaration-for (var) )
       ) #IF ($name-declared-in (detailed-declaration-for
       (var) ) $) is-actually-allocated-in-entry;

    => overlay-element-med-packed-rel-addr
       (first-overlay-mention-of (name-declared-in
       (detailed-declaration-for (var) ) ) ) #IF
       ($name-declared-in (detailed-declaration-for (var) )
       $) is-mentioned-in-a-subordinate-ovly-dec #.
```

```
#DF is-densely-packed (dec)

     "{ ($dec$) is-table-declaration }"

     => packing-spec-of (dec) #EQW 'D' #.


#DF dense-packed-item-entry-relative-addr (var)

     "{ ($var$) is-ordinary-item-reference }"

     => entry-rel-addr-of-dense-allocated-name
        (name-declared-in (detailed-declaration-for (var) )
        ) #IF ($name-declared-in (detailed-declaration-for
        (var) ) $) is-actually-allocated-in-entry;

     => overlay-element-dense-rel-addr
        (first-overlay-mention-of (name-declared-in
        (detailed-declaration-for (var) ) ) ) #IF
        ($name-declared-in (detailed-declaration-for (var) )
        $) is-mentioned-in-a-subordinate-ovly-dec #.


#DF is-string-item-reference (nx)

     => detailed-declaration-for (nx) #IS
        <string-item-declaration> #.


#DF indexed-string-item-relative-address (var, index-vals)

     => ($entry-rel-addr-of-bead ( #FIRST-ELEMENT-IN
        index-vals, "of" var) , "by" 2 #TH-ELEMENT-IN
        index-vals, "and" number-of-words-per-entry-in
        (table-implicitly-referenced-by (var) ) $)
        modified-for-serial-table #IF #NOT
        ($table-implicitly-referenced-by (var) $)
        is-parallel-table #.


#DF entry-rel-addr-of-bead (n, "of" var)

     => ($basic-string-item-addr (detailed-declaration-for
        (var) ) , "for" n "beads"$) adjusted-to-nth-bead #.
```

```
#DF basic-string-item-addr (dec)

     "{ dec #IS <string-item-declaration> }"

     => create-standard-reference-address
        (first-bead-field-descriptor (dec),
        next-bead-field-descriptor (dec),
        beads-per-word-from (dec) ) #.


#DF beads-per-word-from (dec)

     "{ dec #IS <string-item-declaration> }"

     => #SEG 15 #OF dec #.


#DF first-bead-field-descriptor (dec)

     "{ dec #IS <string-item-declaration> }"

     => build-field-descriptor (word-pos-from (dec),
        bit-pos-from (dec), nr-of-bits-in-field-from
        (basic-addr-from-item-descriptor
        (typed-item-description-of (dec) ) ) ) #.


#DF next-bead-field-descriptor (dec)

     "{ dec #IS <string-item-declaration> }"

     => next-packed-bead-field-descriptor (dec) #IF ($dec$)
        is-packed-string-declaration;

     => next-unpacked-bead-field-descriptor (dec) #IF
        ($dec$) is-unpacked-string-declaration;

     => next-med-packed-bead-field-descriptor (dec) #IF
        ($dec$) is-med-packed-string-declaration #.


#DF is-packed-string-declaration (dec)

     "{dec #IS <string-item-declaration>}"

     => #TRUE #IF optional-packing-specification-of (dec)
        #EQW #NIL;
```

```
      => #TRUE #IFF optional-packing-specification-of (dec)
         #EQW 'D' #OTHERWISE #.


#DF optional-packing-specification-of (dec)

      "{ dec #IS <string-item-declaration> #U
      <like-table-declaration> #U
      <ordinary-table-declaration> #U
      <defined-entry-item-declaration>}"

      => #SEG 9 #OF dec #IF dec #IS <like-table-declaration>
         #U <ordinary-table-declaration> ;

      => #SEG 11 #OF dec #OTHERWISE #.


#DF next-packed-bead-field-descriptor (dec)

      "{dec #IS <string-item-declaration>}"

      => build-next-field-descriptor ( nr-of-bits-in-field-of
         ( first-bead-field-descriptor (dec) ) ,
         field-first-bit-of (first-bead-field-descriptor
         (dec) ) , words-between-beads-from (dec) ,
         beads-per-word-from (dec) ) #.


#DF words-between-beads-from (dec)

      "{dec #IS <string-item-declaration>}"

      => #SEG 13 #OF dec #.


#DF is-unpacked-string-declaration (dec)

      "{dec #IS <string-item-declaration>}"

      => #TRUE #IFF optional-packing-specification-of (dec)
         #EQW 'N' #.


#DF next-unpacked-bead-field-descriptor (dec)

      "{dec #IS <string-item-declaration>}"

      => build-next-field-descriptor (bits-per-word,
```

```
      field-first-bit-of (first-bead-field-descriptor
      (dec)), words-between-beads-from (dec), 1) #.


#DF is-med-packed-string-declaration (dec)

    "{dec #IS <string-item-declaration>}"

    => #TRUE #IFF optional-packing-specification-of (dec)
       #EQW 'M' #.


#DF next-med-packed-bead-field-descriptor (dec)

    "{dec #IS <string-item-declaration>}"

    => build-next-field-descriptor (bits-per-byte,
       field-first-bit-of (first-bead-field-descriptor
       (dec)), words-between-beads-from (dec),
       beads-per-word-from (dec)) #.


#DF adjusted-to-nth-bead (s-ref-addr,n)

    "{n>=0 #AND ($s-ref-addr$) is-bead-reference-address}"

    =>s-ref-addr #IF n=0;

    => ($ ($s-ref-addr$) adjusted-to-next-bead, "to" n -
       1$) adjusted-to-nth-bead #OTHERWISE #.


#DF adjusted-to-next-bead (s-ref-addr)

    => ($($ s-ref-addr, "with"
       successor-field-bead-descriptor-of (s-ref-addr)$)
       replacing-field-descriptor, "and"
       successor-next-field-bead-descriptor-of
       (s-ref-addr)$) replacing-next-field-descriptor #.


#DF successor-field-bead-descriptor-of (s-ref-addr)

    => build-field-descriptor (word-address-from
       (s-ref-addr), next-field-first-bit-from
       (s-ref-addr), nr-of-bits-in-field-of (s-ref-addr))
       #IF nr-of-beads-left-in-word-from (s-ref-addr) >1;
```

```
    => build-field-descriptor (next-word-address-from
       (s-ref-addr), first-field-first-bit-from
       (s-ref-addr), nr-of-bits-in-field-of (s-ref-addr))
       #OTHERWISE #.


#DF successor-next-field-bead-descriptor-of (s-ref-addr)

    => next-field-descriptor-of (s-ref-addr) #IF
       nr-of-beads-left-in-word-from (s-ref-addr) >1;

    => ($next-field-descriptor-of (s-ref-addr), "with"
       nr-of-beads-per-word-of (s-ref-addr)$)
       replacing-nr-of-beads-left-in-word #OTHERWISE #.


#DF replacing-nr-of-beads-left-in-word (s-ref-addr,n)

    => ($s-ref-addr, "with" ($ next-field-descriptor-of
       (s-ref-addr), "with" n$) replacing-nr-of-beads-left
       $) replacing-next-field-descriptor #.


#DF replacing-nr-of-beads-left (nxt-fld-desc, n)

    => build-next-field-descriptor (
       bits-between-field-first-bits-of (nxt-fld-desc),
       first-field-first-bit-of (nxt-fld-desc),
       words-between-fields-of (nxt-fld-desc), n) #.


#DF nr-of-beads-per-word-of (s-ref-addr)

    => 3 #TH-ELEMENT-IN (s-ref-addr) #.


#DF nr-of-beads-left-in-word-from (s-ref-addr)

    => nr-of-beads-left-in-word-of
       (next-field-descriptor-of (s-ref-addr)) #.


#DF nr-of-beads-left-in-word-of (nxt-fld-desc)

    => 4 #TH-ELEMENT-IN nxt-fld-desc #.


#DF is-defined-entry-item-reference (nx)
```

```
    => detailed-declaration-for (nx) #IS
       <defined-entry-item-declaration> #.

#DF indexed-defined-item-relative-address (var, index-vals)

    "{ ($var$) is-defined-entry-item-reference}"

    => ($defined-item-entry-relative-addr (var), "by"
       #FIRST-ELEMENT-IN (index-vals) , "and"
       number-of-entries-of (table-implicitly-referenced-by
       (var))$) modified-for-parallel-table #IF
       ($table-implicitly-referenced-by (var) $)
       is-parallel-table;

    => ($defined-item-entry-relative-addr (var) , "by"
       #FIRST-ELEMENT-IN (index-vals) , "and"
       number-of-words-per-entry-in
       (table-implicitly-referenced-by (var)) $)
       modified-for-serial-table #OTHERWISE #.

#DF defined-item-entry-relative-addr (var)

    "{ ($var$) is-defined-entry-item-reference}"

    => shifted-ref-addr (basic-addr-from-item-descriptor
       (item-description-of (detailed-declaration-for
       (var))), "to" bit-addr-of-defined-item-of
       (detailed-declaration-for (var))) #.

#DF bit-addr-of-defined-item-of (dec)

    "{dec #IS <defined-entry-item-declaration>}"

    => bit-addr-built-from (word-pos-from (dec),
       bit-pos-from (dec)) #.

#DF word-pos-from (dec)

    "{dec #IS <string-item-declaration> #U
    <defined-entry-item-declaration>}"

    => #CONVERT 10 (#STRING-OF-TERMINALS-OF ( #SEG 7 #OF
       dec)) #.
```

```
#DF bit-pos-from (dec)

     "{dec #IS <string-item-declaration> #U
     <defined-entry-item-declaration>}"

     => #CONVERT 10 ( #STRING-OF-TERMINALS-OF ( #SEG 9 #OF
        dec)) #.


#DF indexed-array-element-relative-address (var,index-vals)

     => indexed-boolean-array-elt-addr (var, index-vals) #IF
        type (var) #EQW 'boolean';

     => ($basic-array-element-addr (detailed-declaration-for
        (var)) , "with" index-offset (var, "by" index-vals)
        $) replacing-first-word-address #OTHERWISE #.


#DF index-offset (var,index-vals)

     => index-map (index-vals, "wrt" dimension-bounds-from
        (detailed-declaration-for (var))) #.


#DF index-map (ix-list, "wrt" bounds-list)

     "{ #FOR-ALL x #IN ix-list #CS bounds-list
     #IT-IS-TRUE-THAT (x #IS #INTEGER) #AND #LENGTH
     (ix-list) = #LENGTH (bounds-list)}"

     => 0 #IF #LENGTH (ix-list) =0;

     => (#LAST-ELEMENT-IN ix-list) * dimension-product
        (all-but-last-element-in (bounds-list)) + index-map
        (all-but-last-element-in (ix-list), "wrt"
        all-but-last-element-in (bounds-list) ) #OTHERWISE
        #.


#DF all-but-last-element-in (seq)

     "{ seq #IS #SEQUENCE #AND #LENGTH (seq) > 0 }"

     => #INITIAL-SUBSEQ-OF-LENGTH (#LENGTH (seq) - 1) #OF
        seq #.
```

```
#DF dimension-product (bounds-list)

    "{ #FOR-ALL x #IN bounds-list #IT-IS-TRUE-THAT (x #IS
    #INTEGER) }"

    => 1 #IF #LENGTH (bounds-list) =0;

    => (1 + (#FIRST-ELEMENT-IN bounds-list) ) *
       dimension-product (all-but-first-element-in
       (bounds-list)) #OTHERWISE #.


#DF indexed-boolean-array-elt-addr (var,index-vals)

    "{ ($var$) is-array-reference #AND type (var) #EQW
    'boolean'}"

    => ($($basic-addr-of-boolean-array-column-from
       (detailed-declaration-for (var)) , "with"index-map
       (all-but-first-element-in (index-vals) , "wrt"
       all-but-first-element-in (dimension-bounds-from
       (detailed-declaration-for (var)))) $)
       replacing-first-word-address, "at" #FIRST-ELEMENT-IN
       index-vals$) taken-as-the-boolean-array-elt-posn #.


#DF taken-as-the-boolean-array-elt-posn (s-ref-addr, n)

    "{n>=o}"

    => ($ ($ ($s-ref-addr, "to"n$)
       adjusted-reference-address, "with" 1$)
       replacing-nr-of-bits, "and" 1$)
       replacing-bits-in-field #.


#DF replacing-bits-in-field (s-ref-addr, n)

    => ($s-ref-addr, "with" ($ field-descriptor-of
       (s-ref-addr), "with"n$) replacing-fld-length$)
       replacing-field-descriptor #.


#DF simple-standard-reference-address (nx)

    "{nx #IS <simple-variable> #U <name> #U
```

```
    <loop-variable>}"

    => ref-addr-of-allocating-instance-of (nx) #.


#DF ref-addr-of-allocating-instance-of (nx)

    "{nx #IS <simple-variable> #U <name> #U
    <loop-variable>}"

    => allocating-instance-reference-addr-of (nx, "in"
       addressing-unit-containing (nx)) #.


#DF allocating-instance-reference-addr-of (nx, "in" adunit)

    "{ nx #IS<simple-variable> .u<name> #U <loop-variable>
    #AND ($adunit$) is-addressing-unit}"

    => ($relative-addr (allocating-instance-of (nx, "in"
       adunit)), "with" first-word-addr-of-addressing-unit
       (addressing-unit-containing (allocating-instance-of
       (nx, "in" adunit)))$) added-to-first-word-address #.


#DF allocating-instance-of (nx, "in" adunit)

    "{nx #IS <simple-variable> #U <name> #U <loop-variable>
    #AND ($adunit$) is-addressing-unit}"

    => first-allocated-name-for (nx, "in" adunit) #IF ($nx,
       "in" adunit$) refers-to-an-allocated-name;

    => allocating-instance-of (nx, "in"
       addressing-unit-containing (declaration-for (nx)))
       #IF ($nx$) references-a-common-variable #AND ($nx$)
       is-not-in-common;

    => allocating-overlay-instance-of (nx) #IF ($nx$)
       references-an-overlay-variable #.


#DF first-allocated-name-for (nx, "in" adunit)

    "{ ($adunit$) is-an-addressing-unit nx #IS
    <simple-variable> #U <name> #U <loop-variable>}"

    => #FIRST alloc-name #IN sequence-of-allocated-names-in
```

```
      (adunit) #SUCH-THAT (($nx, "and" alloc-name$)
      designate-the-same-allocated-variable) #.

#DF designate-the-same-allocated-variable (nx, "and"
alloc-name)

    "{nx #IS <simple-variable> #U <name> #U
    <loop-variable>}"

    => #STRING-OF-TERMINALS-OF (nx) #EQW
       #STRING-OF-TERMINALS-OF (alloc-name) #IF nx #IS
       <loop-variable>;

    => implied-declaration-for (alloc-name) #EQN
       declaration-for (nx) #IF ($alloc-name$)
       implicitly-first-refers-to-a-table;

    => #TRUE #IFF #STRING-OF-TERMINALS-OF (nx) #EQW
       #STRING-OF-TERMINALS-OF (alloc-name) #AND
       declaration-for (alloc-name) #EQN declaration-for
       (nx) #OTHERWISE #.


#DF refers-to-an-allocated-name (nx, "in" adunit)

    "{nx #IS <simple-variable> #U <name> #U <loop-variable>
    #AND ($adunit$) is-addressing-unit}"

    => #TRUE #IFF #THERE-EXISTS alloc-name #IN
       sequence-of-allocated-names-in (adunit) #SUCH-THAT (
       ($ nx, "and" alloc-name$)
       designate-the-same-allocated-variable) #.


#DF allocating-overlay-instance-of (nx)

    "{ ($nx$) references-an-overlay-variable #AND nx #IS
    <simple-variable> #U <name> #AND ($adunit$)
    is-addressing-unit}"

    => first-mention-of (nx, "in" first-ovly-dec-mentioning
       (nx)) #.


#DF first-ovly-dec-mentioning (nx)

    "{ ($nx$) references-an-overlay-variable}"
```

```
    => #FIRST-ELEMENT-IN
       sequence-of-overlay-declarations-mentioning (nx) #.


#DF first-mention-of (nx, "in" ovly-dec)

    "{ovly-dec #IS <independent-overlay-declaration>}"

    => #FIRST ovly-name #IN (#SEQUENCE-OF <name> #IN
       ovly-dec) #SUCH-THAT ( ($nx, ovly-dec$) match) #.


#DF relative-addr (nx)

    "{nx #IS <name> #U <loop-variable>}"

    => relative-addr-of-allocated-name (nx) #IF ($nx$)
       is-actually-to-be-allocated;

    => overlay-element-rel-addr (nx) #IF ($nx$)
       references-an-overlay-variable #.


#DF added-to-first-word-address (s-ref-addr, n)

    "{ ($s-ref-addr$) is-standard-reference-address #AND
    n>=o}"

    => ($s-ref-addr, "with" word-address-from (s-ref-addr)
       + n$) replacing-first-word-address #.


#DF entry-standard-reference-address (nx)

    "{nx #IS <entry-variable>}"

    => ($ indexed-entry-standard-relative-address
       (table-name-given-in (nx), index-values
       (index-given-in (nx))), "with" word-address-from
       (simple-standard-reference-address
       (table-name-given-in (nx)))$)
       added-to-first-word-address #.


#DF table-name-given-in (nx)

    "{nx #IS <entry-variable>}"
```

```
    => #SEG 1 #OF (#SEG 5 #OF nx) #.


#DF index-given-in (nx)

    "{nx #IS <entry-variable>}"

    => #SEG 9 #OF nx #.


#DF indexed-entry-standard-relative-address (tab-name,
ix-vals)

    => contiguous-word-ref-addr-at ( #FIRST-ELEMENT-IN
       ix-vals * nr-wds-per-entry-in (tab-name), "with"
       nr-wds-per-entry-in (tab-name) * bits-per-word
       "bits") #.


#DF nr-wds-per-entry-in (tab-name)

    => number-of-words-per-entry-in
       (table-implicitly-referenced-by (tab-name)) #IF
       ($tab-name$) references-a-table-item;

    => number-of-words-per-entry-in (declaration-for
       (tab-name)) #OTHERWISE #.


#DF special-int-var-reference-address (sp-int-var)

    "{sp-int-var #IS <special-integer-variable>}"

    => standard-reference-address-of
       (loop-variable-constituting (sp-int-var)) #IF
       ($sp-int-var$) is-loop-variable;

    => bit-modifier-reference-using (index-values
       (index-list-of (sp-int-var)),

    "and" standard-reference-address-of (object-variable-of
    (sp-int-var)))
       #IF ($sp-int-var$) is-bit-functional-modifier;

    => char-modifier-reference-using
       (standard-reference-address-of (object-variable-of
       (sp-int-var))) #IF ($sp-int-var$)
```

```
     is-char-functional-modifier;

  => pos-modifier-reference-using (file-name-of
     (sp-int-var)) #IF ($sp-int-var$)
     is-pos-functional-modifier;

  => nent-modifier-reference-of (sp-int-var) #IF
     ($sp-int-var$) is-nent-functional-modifier #.


#DF loop-variable-constituting (sp-int-var)

  "{sp-int-var #IS <special-integer-variable> #AND
  ($sp-int-var$) is-loop-variable}"

  => #SEG 1 #OF sp-int-var #.


#DF bit-modifier-reference-using (ix-vals,s-ref-addr)

  => ($ ($ s-ref-addr, "to"
     first-bit-position-indicated-in (ix-vals) $)
     adjusted-reference-address, "with"
     nr-of-bits-indicated-in (ix-vals) $)
     replacing-nr-of-bits #.


#DF first-bit-position-indicated-in (ix-vals)

  => #FIRST-ELEMENT-IN ix-vals #.


#DF nr-of-bits-indicated-in (ix-vals)

  => 1 #IF #LENGTH (ix-vals) <2;

  => 2 #TH-ELEMENT-IN ix-vals #OTHERWISE #.


#DF adjusted-reference-address (s-ref-addr, "to" n)

  => ($s-ref-addr, "with" n$)
     bits-dropped-from-first-field #IF n <
     nr-of-bits-in-field-from (s-ref-addr);

  => ($ ($s-ref-addr$) adjusted-to-next-field, "to" n -
     nr-of-bits-in-field-from (s-ref-addr) $)
     adjusted-reference-address #OTHERWISE #.
```

```
#DF bits-dropped-from-first-field (s-ref-addr, n)

    => create-standard-reference-address (
       ($field-descriptor-of (s-ref-addr) , "by" n$)
       adjusted-field-descriptor, next-field-descriptor-of
       (s-ref-addr) , nr-of-bits-in (s-ref-addr) - n) #.

#DF adjusted-field-descriptor (fld-desc, "by" n)

    => build-field-descriptor (word-address-of (fld-desc) ,
       field-first-bit-of (fld-desc) +n,
       nr-of-bits-in-field-of (fld-desc) - n) #.

#DF char-modifier-reference-using (s-ref-addr)

    => ($s-ref-addr, "with" ($field-descriptor-of
       (s-ref-addr) , "with" bits-in-floating-exponent$)
       replacing-fld-length $) replacing-field-descriptor
       #.

#DF file-name-of (sp-int-var)

     "{ ($sp-int-var$) is-pos-functional-modifier}"

    => #SEG 5 #OF sp-int-var #.

#DF pos-modifier-reference-using (fname)

    => #UNDEFINED #.

#DF nent-modifier-reference-of (sp-int-var)

    "{ ($sp-int-var$) is-nent-functional-modifier}"

    => create-standard-reference-address
       (nent-word-field-descriptor (sp-int-var),
       null-next-field-descriptor, nent-size (sp-int-var))
       #.

#DF null-next-field-descriptor
```

```
     => build-next-field-descriptor ( #UNDEFINED,
        #UNDEFINED, #UNDEFINED, #UNDEFINED) #.


#DF nent-word-field-descriptor (sp-int-var)

     => build-field-descriptor (word-address-from
        (standard-reference-address-of (table-name-of
        (sp-int-var))) , bits-per-word - nent-size
        (sp-int-var) , nent-size (sp-int-var)) #.


#DF special-fix-var-reference-address (sp-fix-var)

     "{sp-fix-var #IS <special-fixed-variable>}"

     => mant-modifier-reference-using
        (standard-reference-address-of (object-variable-of
        (sp-fix-var))) #.


#DF mant-modifier-reference-using (s-ref-addr)

     => ($ s-ref-addr, "with" ($field-descriptor-of
        (s-ref-addr) , "with" bits-in-floating-mantissa$)
        replacing-fld-length$) replacing-field-descriptor #.


#DF special-literal-var-reference-address (sp-fix-var)

     "{sp-fix-var #IS <special-fixed-variable}"

     => byte-modifier-reference-using (index-values
        (index-list-of (sp-fix-var)) , "and"
        standard-reference-address-of (object-variable-of
        (sp-fix-var))) #.


#DF byte-modifier-reference-using (ix-vals, s-ref-addr)

     => ($ ($ s-ref-addr, "to"
        first-byte-position-indicated-in (ix-vals)
        *bits-per-byte $) adjusted-reference-address, "with"
        nr-of-bytes-indicated-in (ix-vals) *bits-per-byte$)
        replacing-nr-of-bits #.
```

```
#DF first-byte-position-indicated-in (ix-vals)

    => #FIRST-ELEMENT-IN ix-vals #.


#DF nr-of-bytes-indicated-in (ix-vals)

    => 1 #IF #LENGTH (ix-vals) <2;

    => 2 #TH-ELEMENT-IN ix-vals #OTHERWISE #.


#DF special-boolean-var-reference-address (nx)

    "{nx #IS <special-boolean-variable>}"

    => last-bit-ref-addr-of (standard-reference-address-of
       (object-variable-of (nx))) #.


#DF last-bit-ref-addr-of (s-ref-addr)

    => last-bit-ref-addr-of ( ($s-ref-addr$)
       adjusted-to-next-field) #IF nr-of-bits-in
       (s-ref-addr) > nr-of-bits-in-field-from (s-ref-addr)
       ;

    => ($s-ref-addr, "with" nr-of-bits-in (s-ref-addr) -
       1$) bits-dropped-from-first-field #OTHERWISE #.
```

```
#DF sequence-of-allocated-addressing-units-in (sys)

     "{ sys #IS <jovial-j3-system>}"

     => #SUBSEQUENCE-OF-ELEMENTS alloc-adunit #IN
        sequence-of-addressing-units-in (sys) #SUCH-THAT
        (($alloc-adunit$) is-allocated-addressing-unit) #.


#DF is-allocated-addressing-unit (adunit)

     "{ ($adunit$) is-a-true-adunit}"

     => #TRUE #IF adunit #IS <program> #U <close-subprogram>
        #U <procedure-subprogram> ;

     => #TRUE #IF ($adunit$)
        is-first-common-dec-with-its-name #.


#DF is-first-common-dec-with-its-name (dec)

     "{($dec$)
     is-first-common-dec-with-its-name-in-a-compool}"

     => #TRUE #IFF #FOR-ALL other-common #IN
        seq-of-system-common-decs-preceding (dec)
        #IT-IS-TRUE-THAT (optional-common-block-name-of
        (other-common) #NEQW optional-common-block-name-of
        (dec)) #.


#DF seq-of-system-common-decs-preceding (c-dec)

     "{($c-dec$)
     is-first-common-dec-with-its-name-in-a-compool}"

     => #SUBSEQUENCE-OF-ELEMENTS other-common #IN
        sequence-of-addressing-units-in
        (system-containing(c-dec)) #SUCH-THAT (other-common
        #IS <common-declaration> #AND other-common #PRECEDES
        c-dec #IN system-containing (c-dec)) #.


#DF sequence-of-addressing-units-in (sys)
```

```
    "{ sys #IS <jovial-j3-system>}"

    => #SUBSEQUENCE-OF-ELEMENTS adunit #IN
       sequence-of-possible-adunits-in (sys) #SUCH-THAT
       (($adunit$) is-a-true-adunit) #.


#DF sequence-of-possible-adunits-in (sys)

    "{ sys #IS <jovial-j3-system>}"

    => #SEQUENCE-OF <program> #U <close-subprogram> #U
       <procedure-subprogram> #U <common-declaration> #U
       <independent-overlay-declaration> #IN sys #.


#DF is-a-true-adunit (adunit)

    "{ adunit #IS-IN sequence-of-possible-adunits-in
    (system-containing (adunit)}"

    => #TRUE #IF adunit #IS <program> #U <close-subprogram>
       #U <procedure-subprogram>;

    => ($adunit$)
       is-first-common-dec-with-its-name-in-a-compool #IF
       adunit #IS <common-declaration>;

    => ($adunit$) is-absolute-overlay-declaration #IF
       adunit #IS <independent-overlay-declaration> #.


#DF is-first-common-dec-with-its-name-in-a-compool (c-dec)

    "{ c-dec #IS <common-declaration> #AND c-dec #IS-IN
    sequence-of-possible-adunits-in (system-containing
    (c-dec))}"

    => #TRUE #IFF #FOR-ALL other-common #IN
       seq-of-compool-common-decs-preceding (c-dec)
       #IT-IS-TRUE-THAT (optional-common-block-name-of
       (other-common) #NEQW optional-common-block-name-of
       (c-dec )) #.


#DF seq-of-compool-common-decs-preceding (c-dec)

    "{ c-dec #IS <common-declaration>}"
```

```
    => #SUBSEQUENCE-OF-ELEMENTS other-common #IN
       (#SEQUENCE-OF <common-declaration> #IN
       compool-containing (c-dec)) #SUCH-THAT (other-common
       #PRECEDES c-dec #IN compool-containing (c-dec)) #.


#DF compool-containing (nx)

    => #LAST cp #IN ( #SEQUENCE-OF-ANCESTORS-OF nx)
       #SUCH-THAT (cp #IS <implementation-compool>) #.


#DF optional-common-block-name-of (c-dec)

    "{ c-dec #IS <common-declaration>}"

    => #SEG 3 #OF c-dec #.


#DF is-absolute-overlay-declaration (i-o-d)

    "{ i-o-d #IS <independent-overlay-declaration>}"

    => #TRUE #IFF i-o-d #IS #CASE 2 #OF
       <independent-overlay-declaration> #.
```

```
#DF first-word-addr-of-addressing-unit (adunit)

    "{ ($adunit$) is-an-addressing-unit}"

    => first-word-addr-of-allocated-addressing-unit
       (adunit) #IF ($adunit$)
       is-an-allocated-addressing-unit;

    => first-word-addr-of-unallocated-addressing-unit
       (adunit) #IF ($adunit$)
       is-an-unallocated-addressing-unit #.

#DF is-an-unallocated-addressing-unit (adunit)

    => #TRUE #IFF ($adunit$) is-an-addressing-unit #AND
       #NOT ($adunit$) is-an-allocated-addressing-unit #.

#DF is-an-addressing-unit (nx)

    "{ nx #IS #NODE }"

    => #TRUE #IFF nx #IS-IN sequence-of-addressing-units-in
       (system-containing (nx)) #.

#DF first-word-addr-of-unallocated-addressing-unit (adunit)

    "{ ($adunit$) is-unallocated-addressing-unit}"

    => overlay-origin (adunit) #IF ($adunit$)
       is-absolute-overlay-declaration;

    => first-word-addr-of-addressing-unit (
       first-common-dec-with-the-name-of-this (adunit)) #IF
       ($adunit$)
       is-first-common-dec-with-its-name-in-a-compool #.

#DF overlay-origin (adunit)

    "{ ($adunit$) is-absolute-overlay-declaration}"

    => #CONVERT 10 (digits-of (origin-specifier-of
       (adunit))) #.
```

```
#DF digits-of (num)

    "{ num #IS <numeral> #U <octal-constant>}"

    => num #IF num #IS <numeral>;

    => word-between ('O(',"and" ')', "in"
       #STRING-OF-TERMINALS-OF (num)) #IF num #IS
       <octal-constant> #.


#DF origin-specifier-of (ovly-dec)

    "{ ($ovly-dec$) is-absolute-overlay-declaration}"

    => #SEG 3 #OF ovly-dec #.


#DF first-common-dec-with-the-name-of-this (adunit)

    "{($adunit$)
    is-first-common-dec-with-its-name-in-a-compool}"

    => #FIRST other-common #IN
       seq-of-system-common-decs-preceding (adunit)
       #SUCH-THAT (optional-common-block-name-of
       (other-common) #EQW optional-common-block-name-of
       (adunit)) #.


#DF is-an-allocated-addressing-unit (nx)

    "{ nx #IS #NODE}"

    => #TRUE #IFF nx #IS-IN
       sequence-of-allocated-addressing-units-in
       (system-containing (nx)) #.


#DF first-word-addr-of-allocated-addressing-unit (adunit)

    "{($adunit$) is-an-allocated-addressing-unit}"

    => first-word-addr-in-memory #IF adunit #EQ
       #FIRST-ELEMENT-IN
       sequence-of-allocated-addressing-units-in
```

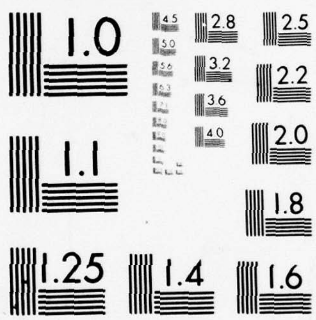
```
        (system-containing (adunit));

     => 1 + last-word-addr-of-addressing-unit
        (addressing-unit-preceding (adunit)) #OTHERWISE #.


#DF first-word-addr-in-memory

     => 0 #.


#DF addressing-unit-preceding (adunit)

     "{ adunit #IS-IN
     sequence-of-allocated-addressing-units-in
     (system-containing (adunit)) }"

     => preceding (adunit ,"in"
        sequence-of-allocated-addressing-units-in
        (system-containing (adunit))) #.


#DF preceding (nx, "in" seq)

     "{ nx #IS-IN seq #AND nx #NEQ #FIRST-ELEMENT-IN seq}"

     => (#ORDPOSIT nx #IN seq - 1) #TH-ELEMENT-IN seq #.


#DF last-word-addr-of-addressing-unit (adunit)

     "{($adunit$) is-an-addressing-unit}"

     => first-word-addr-of-addressing-unit (adunit) +
        last-word-addr
        (relative-addr-of-structure-designated-by (
        last-allocated-name-in (adunit))) #.


#DF last-word-addr (s-ref-addr)

     "{ ($ s-ref-addr$) is-standard-reference-address}"

     => last-word-addr-from-contiguous-word-ref-addr
        (s-ref-addr) #IF ($s-ref-addr$)
        is-a-contiguous-word-ref-addr;

     => word-address-of (#LAST-ELEMENT-IN
```

```
     field-descriptor-sequence-implied-in (s-ref-addr))
     #OTHERWISE #.


#DF last-word-addr-from-contiguous-word-ref-addr
(s-ref-addr)

    "{ ($s-ref-addr$) is-standard-reference-address #AND
    ($s-ref-addr$) is-a-contiguous-word-ref-addr }"

    => word-address-from (s-ref-addr) +
       ((nr-of-bits-in(s-ref-addr) - 1) / bits-per-word) #.


#DF is-a-contiguous-word-ref-addr (s-ref-addr)

    "{ ($s-ref-addr$) is-standard-reference-address}"

    => #TRUE #IFF s-ref-addr #EQ
       contiguous-word-ref-addr-at (word-address-from
       (s-ref-addr), "with" nr-of-bits-in (s-ref-addr)
       "bits") #.


#DF contiguous-word-ref-addr-at (n, "with" m "bits")

    => ($($ basic-contiguous-word-ref-addr, "with" n $)
       replacing-first-word-address, "and" m $)
       replacing-nr-of-bits #.


#DF last-allocated-name-in (adunit)

    "{($adunit$) is-an-allocated-addressing-unit }"

    => #LAST-ELEMENT-IN sequence-of-allocated-names-in
       (adunit) #.


#DF sequence-of-allocated-names-in (adunit)

    "{ ($adunit$) is-an-addressing-unit }"

    => sequence-of-allocated-names-implied-in
       (addressing-unit-parts-from (adunit)) #.


#DF addressing-unit-parts-from (adunit)
```

AD-A049 474 TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CALIF F/G 9/2
SEMANOL (76) SPECIFICATION OF JOVIAL (J3). VOLUME III.(U)
NOV 77 F C BELZ, I M GREEN F30602-76-C-0238
UNCLASSIFIED RADC-TR-77-365-VOL-3 NL

4 OF 4
AD
A049474

END
DATE
FILMED
3-78
DDC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
    "{ ($adunit$) is-an-addressing-unit}"

    => \adunit\ #IF adunit #IS &lt;program&gt; #U
       &lt;close-subprogram&gt; #U &lt;procedure-subprogram&gt;;

    => ovly-decs-in-block-rooted-by (adunit) #IF ($adunit$)
       is-absolute-overlay-declaration;

    => compool-common-decs-with-same-name-as (adunit) #IF
       ($adunit$)
       is-first-common-dec-with-its-name-in-a-compool #.


#DF ovly-decs-in-block-rooted-by (ovly-dec)

    "{ ovly-dec #IS &lt;independent-overlay-declaration&gt;}"

    => overlay-block-derived-by (\ovly-dec\ ,"from"
       overlay-decs-in-adunit-following (ovly-dec)) #.

    "{ ovly-dec #IS &lt;independent-overlay-declaration&gt;}"


#DF overlay-decs-in-adunit-following (ovly-dec)

    "{ ovly-dec #IS &lt;independent-overlay-declaration&gt;}"

    => elements-following (ovly-dec, "in"
       sequence-of-overlay-decs-in-adunit
       (addressing-unit-containing (ovly-dec))) #.


#DF sequence-of-overlay-decs-in-adunit (adunit)

    "{ ($adunit$) is-an-addressing-unit}"

    => #SUBSEQUENCE-OF-ELEMENTS ovly-dec #IN domain
       (adunit) #SUCH-THAT (ovly-dec #IS
       &lt;independent-overlay-declaration&gt;) #.


#DF elements-following (nx, "in" seq)

    "{ nx #IS #NODE #AND seq #IS #SEQUENCE }"

    => #TERMINAL-SUBSEQ-OF-LENGTH (#LENGTH(seq) -
       (#ORDPOSIT nx #IN seq)) #OF seq #.
```

```
#DF overlay-block-derived-by (ovly-block-seq, "from"
candidate-seq)

    "{ #FOR-ALL x #IN ovly-block-seq #CS candidate-seq
    #IT-IS-TRUE-THAT (x #IS
    <independent-overlay-declaration>}"

    => ovly-block-seq #IF candidate-seq #EQ #NILSEQ;

    => overlay-block-derived-by (ovly-block-seq #CS \
       #FIRST-ELEMENT-IN candidate-seq\, "from"
       all-but-first-element-in (candidate-seq)) #IF
       ($#FIRST-ELEMENT-IN candidate-seq, "by"
       ovly-block-seq$) is-a-derived-ovly-dec;

    => overlay-block-derived-by (ovly-block-seq, "from"
       all-but-first-element-in (candidate-seq)) #OTHERWISE
       #.

#DF is-a-derived-ovly-dec ( dec, "by" ovly-block)

    => #FALSE #IF ($dec$) is-absolute-overlay-declaration;

    => #TRUE #IFF #THERE-EXISTS ovly-dec #IN ovly-block
       #SUCH-THAT ( ($overlay-initiator-of (dec), "in"
       ovly-dec$) is-mentioned) #OTHERWISE #.

#DF compool-common-decs-with-same-name-as (adunit)

    "{ ($adunit$)
    is-first-common-dec-with-its-name-in-a-compool}"

    => #SUBSEQUENCE-OF-ELEMENTS comp #IN (#SEQUENCE-OF
       <implementation-compool> #IN compool-containing
       (adunit)) #SUCH-THAT (optional-common-block-name-of
       (comp) #EQW optional-common-block-name-of (adunit))
       #.

#DF sequence-of-allocated-names-implied-in (ad-part-seq)

    => #NILSEQ #IF ad-part-seq #EQ #NILSEQ;

    => \ allocated-names-in (#FIRST-ELEMENT-IN
```

```
        ad-part-seq)\ #CS
        sequence-of-allocated-names-implied-in
        (all-but-first-element-in (ad-part-seq)) #OTHERWISE
        #.


#DF allocated-names-in (adpart)

     "{ ($adpart$) is-an-addressing-unit-part }"

     => #SUBSEQUENCE-OF-ELEMENTS alloc-name #IN
        sequence-of-allocation-names-in (adpart) #SUCH-THAT
        (($alloc-name$) is-actually-to-be-allocated) #.


#DF is-actually-to-be-allocated (nm)

     "{ nm #IS <name> #U <loop-variable> #AND ($nm$)
     is-allocation-name}"

     => #PARENT-NODE (nm) #IS <one-factor-for-clause> #U
        <two-factor-for-clause> #U <complete-for-clause> #IF
        nm #IS <loop-variable>;

     => #FALSE #IF ($ nm $) references-a-common-variable
        #AND ($ nm $) is-not-in-common ;

     => #TRUE #IF ($ nm $)
        implicitly-first-refers-to-a-table ;

     => #FALSE #IF ($nm$) references-a-table-item;

     => #TRUE #IF ($nm$)
        is-allocated-prime-overlay-initiator;

     => #FALSE #IF ($nm$) references-an-overlay-variable;

     => #TRUE #IFF ($nm$)
        is-first-reference-to-a-declared-variable #OTHERWISE
        #.

#DF references-a-common-variable (nm)

     "{nm #IS <name> #AND ($nm$) is-allocation-name}"

     => #TRUE #IFF ($declaration-for (nm)$) is-in-common #.
```

```
#DF is-not-in-common (nm)

     => #NOT is-in-common (nm) #.

#DF is-in-common (nm)

     "{nm #IS <name> #AND ($nm$) is-allocation-name}"

     => addressing-unit-containing (nm) #IS
        <common-declaration> #.

#DF implicitly-first-refers-to-a-table (nm)

     "{nm #IS <name> #AND ($nm$) is-allocation-name}"

     => #FALSE #IF #NOT ($nm$) references-a-table-item;

     => #FALSE #IF any-prior-nm-refers-to-the-table-of (nm);

     => #TRUE #OTHERWISE #.

#DF any-prior-nm-refers-to-the-table-of (nm)

     "{ ($nm$) is-allocation-name #AND ($nm$)
     references-a-table-item}"

     => #TRUE #IF #THERE-EXISTS alloc-name #IN
        sequence-of-allocation-names-in
        (addressing-unit-containing (nm)) #SUCH-THAT
        (($table-implicitly-referenced-by (nm), "by"
        alloc-name$) is-explicitly-named);

     => #TRUE #IFF #THERE-EXISTS alloc-name #IN
        elements-preceding (nm, "in"
        sequence-of-allocation-names-in
        (addressing-unit-containing (nm))) #SUCH-THAT
        (($alloc-name, "and" nm$)
        implicitly-refer-to-same-table) #OTHERWISE #.

#DF elements-preceding (x ,"in" seq)

     "{seq #IS #SEQUENCE #AND x #IS-IN seq }"
```

```
    => #INITIAL-SUBSEQ-OF-LENGTH ((#ORDPOSIT x #IN seq) -
       1) #OF seq #.


#DF table-implicitly-referenced-by (nm)

    "{ ($nm$) references-a-table-item}"

    => declaration-for (nm) #IF ($nm$) is-like-declared;

    => table-containing (declaration-for (nm)) #OTHERWISE
       #.


#DF table-containing (nx)

    "{nx #IS #NODE }"

    => #LAST ancestor #IN (#SEQUENCE-OF-ANCESTORS-OF (nx))
       #SUCH-THAT ( ancestor #IS
       <ordinary-table-declaration> #U
       <defined-entry-table-declaration> #U
       <like-table-declaration> ) #.


#DF is-explicitly-named (tab, "by" nm)

    => nm #EQW name-declared-in (tab) #IF declaration-for
       (nm) #EQN tab;

    => #FALSE #OTHERWISE #.


#DF implicitly-refer-to-same-table (nm-one, nm-two)

    "{ ($nm-one$) is-an-allocation-name #AND ($nm-two$)
    is-an-allocation-name #AND ($nm-two$)
    references-a-table-item}"

    => #FALSE #IF #NOT ($nm-one$) references-a-table-item;

    => #TRUE #IFF table-implicitly-referenced-by (nm-one)
       #EQN table-implicitly-referenced-by (nm-two)
       #OTHERWISE #.


#DF sequence-of-allocation-names-in (adpart)
```

```
    "{ ($adpart$) is-an-addressing-unit-part }"

    => #SUBSEQUENCE-OF-ELEMENTS alloc-name #IN
       sequence-of-possible-allocation-names-in (adpart)
       #SUCH-THAT (($alloc-name$) is-allocation-name) #.


#DF sequence-of-possible-allocation-names-in (adpart)

    "{ ($adpart$) is-an-addressing-unit-part }"

    => #SEQUENCE-OF <name> #U <loop-variable> #IN adpart #.


#DF is-allocation-name (nm)

    "{ nm #IS <name> #U <loop-variable>}"

    => #TRUE #IF nm #IS <loop-variable>;

    => a-category-3-declaration-exists-for (nm) #IF
       #PARENT-NODE (nm) #IS <function-name> #U
       <input-operand> #U <loc-name> #U <simple-variable>
       #U <table-name> #U <tabular-name>;

    => #FALSE #OTHERWISE #.


#DF references-a-table-item (nm)

    "{ nm #IS <name> #AND ($nm$) is-allocation-name }"

    => #TRUE #IFF ($nm$) is-ordinary-item-reference #OR
       ($nm$) is-string-item-reference #OR ($nm$)
       is-defined-entry-item-reference #.


#DF is-allocated-prime-overlay-initiator (nm)

    "{ ($nm$) is-allocation-name}"

    => ($nm$) is-prime-initiator #IF ($nm$)
       is-overlay-initiator;

    => #FALSE #OTHERWISE #.


#DF is-prime-overlay-initiator (alloc-name)
```

```
     "{ ($alloc-name$) is-allocation-name }"

     => ($alloc-name$) is-prime-initiator #IF ($alloc-name$)
        is-overlay-initiator;

     => #FALSE #OTHERWISE #.


#DF is-overlay-initiator (alloc-name)

     "{ ($alloc-name$) is-allocation-name}"

     => #TRUE #IFF #THERE-EXISTS ovly-dec #IN
        sequence-of-overlay-declarations-mentioning
        (alloc-name) #SUCH-THAT (($alloc-name, "in"
        ovly-dec$) appears-as-the-overlay-initiator) #.


#DF appears-as-the-overlay-initiator (nm, "m" ovly-dec)

     "{ ovly-dec #IS <independent-overlay-declaration> #AND
     name #IS <name> }"

     => #FALSE #IF ($ovly-dec$)
        is-absolute-overlay-declaration;

     => #TRUE #IFF nm #EQN overlay-initiator-of (ovly-dec)
        #OTHERWISE #.


#DF overlay-initiator-of (ovly-dec)

     "{ ($ovly-dec$) is-overlay-declaration}"

     => data-sequence-initiator-of (first-data-sequence-of
        (overlay-specification-of (ovly-dec))) #.


#DF data-sequence-initiator-of (ds)

     "{ ds #IS <data-sequence> }"

     => #SEG 1 #OF (#SEG 1 #OF ds) #.


#DF overlay-specification-of (ovly-dec)
```

```
     "{($ovly-dec$) is-overlay-declaration}"

     => #SEG 3 #OF ovly-dec #.


#DF first-data-sequence-of (ovly-spec)

     "{ ovly-spec #IS <overlay-specification>}"

     => #SEG 1 #OF ovly-spec #.


#DF is-prime-initiator (nm)

     "{ ($nm$) is-overlay-initiator}"

     => nm #EQN overlay-initiator-of (#FIRST-ELEMENT-IN
        sequence-of-overlay-declarations-mentioning (nm))
        #IF #FOR-ALL ovly-dec #IN
        sequence-of-overlay-declarations-mentioning (nm)
        #IT-IS-TRUE-THAT ( ($nm, "in" ovly-dec $)
        matches-the-overlay-initiator);

     => #FALSE #OTHERWISE #.


#DF matches-the-overlay-initiator (nm, "in" ovly-dec)

     "{ nm #IS <name> #AND ($nm$) is-overlay-declaration}"

     => #FALSE #IF ($ovly-dec$)
        is-absolute-overlay-declaration;

     => #FALSE #IF nm #NEQW overlay-initiator-of (ovly-dec);

     => #TRUE #IFF declaration-for (nm) #EQW declaration-for
        (overlay-initiator-of(ovly-dec)) #OTHERWISE #.


#DF references-an-overlay-variable (nm)

     "{ nm #IS <name> #AND ($nm$) is-an-allocation-name}"

     => #TRUE #IFF
        sequence-of-overlay-declarations-mentioning (nm)
        #NEQS #NILSEQ #.
```

```
#DF sequence-of-overlay-declarations-mentioning (nm)

    "{ nm #IS <name> #AND ($nm$) is-an-allocation-name}"

    => #SUBSEQUENCE-OF-ELEMENTS ovly-dec #IN
       sequence-of-ovly-decs-possibly-mentioning (nm)
       #SUCH-THAT (($nm, "in" ovly-dec$) is-mentioned) #.


#DF sequence-of-ovly-decs-possibly-mentioning (nm)

    "{ nm #IS <name> #AND ($nm$) is-an-allocation-name }"

    => #SEQUENCE-OF <independent-overlay-declaration> #IN
       actual (program-context (declaration-for (nm))) #IF
       actual (program-context (nm)) #IS #NODE-IN actual
       (program-context (declaration-for (nm))) ;

    => #SEQUENCE-OF <independent-overlay-declaration> #IN
       actual (program-context (declaration-for (nm))) #CS
       #SEQUENCE-OF <independent-overlay-declaration> #IN
       actual (program-context (nm)) #OTHERWISE #.


#DF is-mentioned (nm, "in" ovly-dec)

    "{ nm #IS <name> #AND ($nm$) is-an-allocation-name #AND
    ($ovly-dec$) is-overlay-declaration}"

    => #TRUE #IFF #THERE-EXISTS ovly-name #IN (#SEQUENCE-OF
       <name> #IN ovly-dec) #SUCH-THAT ( ($nm,ovly-name$)
       match) #.

#DF match (nm,ovly-nm)

    => #FALSE #IF nm #NEQW ovly-nm;

    => #TRUE #IFF declaration-for (nm) #EQN declaration-for
       (ovly-nm) #OTHERWISE #.


#DF is-first-reference-to-a-declared-variable (nm)

    "{nm #IS <name> #AND ($nm$) is-allocation-name}"

    => #TRUE #IFF #FOR-ALL alloc-name #IN
       sequence-of-allocation-names-in
```

```
     (addressing-unit-containing (nm)) #IT-IS-TRUE-THAT
     (declaration-for (nm) #EQN declaration-for
     (alloc-name) #IMPLIES nm #PRECEDES alloc-name #IN
     sequence-of-allocation-names-in
     (addressing-unit-containing (nm))) #.

#DF addressing-unit-containing (nx)

    "{ nx #IS #NODE}"

    => #FIRST adunit #IN sequence-of-addressing-units-in
       (system-containing (nx)) #SUCH-THAT (nx #IS-IN
       domain (adunit)) #.

#DF domain (adunit)

    "{ ($adunit$) is-an-addressing-unit }"

    => sequence-of-nodes-implied-in
       (addressing-unit-parts-from (adunit)) #.


#DF sequence-of-nodes-implied-in (ad-part-seq)

    "{ #FOR-ALL adpart #IN ad-part-seq #IT-IS-TRUE-THAT
    (($adpart$) is-an-addressing-unit-part)}"

    => #NILSEQ #IF ad-part-seq #EQ #NILSEQ;

    => \ #SEQUENCE-OF-NODES-IN (#FIRST-ELEMENT-IN
       ad-part-seq)\ #CS sequence-of-nodes-implied-in
       (ad-part-seq) #OTHERWISE #.
```

```
#DF relative-addr-of-structure-designated-by (alloc-name)

     "{ ($alloc-name$) is-allocated-name}"

     => relative-addr-of-overlay-block-initiated-by
        (alloc-name) #IF ($alloc-name$)
        is-prime-overlay-initiator;

     => relative-addr-of-allocated-name (alloc-name)
        #OTHERWISE #.


#DF relative-addr-of-overlay-block-initiated-by (alloc-name)

     "{ ($alloc-name$) is-allocated-name #AND ($alloc-name$)
     is-prime-overlay-initiator }"

     => contiguous-word-ref-addr-at (word-address-from
        (relative-addr-of-allocated-name (alloc-name)),
        "with" length-of-overlay-block-initiated-by
        (alloc-name) * bits-per-word "bits") #.


#DF length-of-overlay-block-initiated-by (alloc-name)

     "{ ($ alloc-name $) is-prime-overlay-initiator }"

     => 1 + last-word-addr-in-overlay-block-initiated-by
        (alloc-name) - word-address-from
        (relative-addr-of-allocated-name (alloc-name) ) #.


#DF last-word-addr-in-overlay-block-initiated-by
(alloc-name)

     => maximal-addr-of
        (sequence-of-data-sequence-terminators-from
        (ovly-decs-in-block-initiated-by (alloc-name))) #.


#DF ovly-decs-in-block-initiated-by (alloc-name)

     "{ ($alloc-name$) is-prime-overlay-initiator}"

     => ovly-decs-in-block-rooted-by
        (ind-ovly-dec-containing (alloc-name)) #.
```

```
#DF ind-ovly-dec-containing (alloc-name)

     "{ ($alloc-name$) is-prime-overlay-initiator}"

     => #LAST iod #IN (#SEQUENCE-OF-ANCESTORS-OF alloc-name)
        #SUCH-THAT (iod #IS
        <independent-overlay-declaration>) #.


#DF sequence-of-data-sequence-terminators-from (ovly-block)

     "{ #FOR-ALL ovly-dec #IN ovly-block #IT-IS-TRUE-THAT (
     ($ ovly-dec $) is-overlay-declaration ) }"

     => #NILSEQ #IF ovly-block #EQ #NILSEQ;

     => data-seq-terminator-seq (#FIRST-ELEMENT-IN
        ovly-block) #CS
        sequence-of-data-sequence-terminators-from
        (all-but-first-element-in (ovly-block)) #OTHERWISE
        #.


#DF data-seq-terminator-seq (ovly-dec)

     "{ ($ovly-dec$) is-overlay-declaration}"

     => #SUBSEQUENCE-OF-ELEMENTS term #IN (#SEQUENCE-OF
        <name> #IN ovly-dec) #SUCH-THAT ( ($term$)
        is-data-seq-terminator) #.


#DF is-data-seq-terminator (nm)

     "{ nm #IS <name> }"

     => #TRUE #IFF #PARENT-NODE (#PARENT-NODE(nm)) #IS #CASE
        1 #OF <data-sequence> #.


#DF maximal-addr-of (ds-term-seq)

     "{ #FOR-ALL x #IN ds-term-seq #IT-IS-TRUE-THAT (($x$)
     is-data-seq-terminator}"

     => last-word-addr (overlay-element-rel-addr
```

```
        (#FIRST-ELEMENT-IN ds-term-seq)) #IF #LENGTH
        (ds-term-seq) =1;

     => maximum (last-word-addr (overlay-element-rel-addr
        (#FIRST-ELEMENT-IN ds-term-seq)) , maximal-addr-of
        (all-but-first-element-in (ds-term-seq))) #OTHERWISE
        #.


#DF overlay-element-rel-addr (nm)

     "{ nm #IS <name> #AND ($nm$)
     is-in-overlay-declaration}"

     => overlay-initiator-rel-addr (nm) #IF ($nm$)
        initiates-its-overlay;

     => data-sequence-initiator-rel-addr (nm) #IF ($nm$)
        initiates-its-data-sequence;

     => alloc-name-addr-for (nm ,"at" 1 + last-word-addr
        (overlay-element-rel-addr ( ($nm$)
        preceding-this-nm-in-its-data-sequence))) #OTHERWISE
        #.


#DF initiates-its-overlay (nm)

     "{ nm #IS <name> #AND ($nm$)
     is-in-overlay-declaration}"

     => nm #EQ overlay-initiator-of
        (overlay-declaration-containing (nm)) #IF #NOT
        ($overlay-declaration-containing (nm)$)
        is-absolute-overlay-declaration;

     => nm #EQ first-name-in-overlay
        (overlay-declaration-containing (nm)) #OTHERWISE #.


#DF overlay-initiator-rel-addr (nm)

     "{ ($nm$) initiates-its-overlay}"

     => subordinate-overlay-initiator-rel-addr (nm) #IF
        overlay-declaration-containing (nm) #IS
        <subordinate-overlay-declaration>;
```

```
     => alloc-name-addr-for ( nm, "at" 0 ) #IF
        ($overlay-declaration-containing (nm)$)
        is-absolute-overlay-declaration;

     => prime-overlay-initiator-rel-addr (nm) #IF ($nm$)
        is-prime-overlay-initiator;

     => overlay-element-rel-addr
        (nearest-prior-overlay-reference-to (nm)) #OTHERWISE
        #.


#DF subordinate-overlay-initiator-rel-addr (nm)

     "{ ($nm$) initiates-its-overlay #AND ($nm$)
     is-contained-in-a-subordinate-ovly-dec}"

     => entry-rel-addr-of-unpacked-allocated-name (nm) #IF
        ($nm$) is-prime-subordinate-overlay-initiator;

     => overlay-element-rel-addr
        (nearest-prior-subord-ovly-reference-to (nm))
        #OTHERWISE #.


#DF nearest-prior-subord-ovly-reference-to (nm)

     "{ ($nm$) is-contained-in-a-subordinate-ovly-dec #AND
     ($nm$) initiates-its-overlay #AND #NOT ($nm$)
     is-prime-subordinate-overlay-initiator}"

     => overlay-reference-to (nm, "in" preceding
        (overlay-declaration-containing (nm), "in"
        seq-of-subord-overlay-declarations-mentioning (nm)))
        #.


#DF prime-overlay-initiator-rel-addr (nm)

     "{ ($nm$) is-prime-overlay-initiator}"

     => relative-addr-of-allocated-name (nm) #IF ($nm$)
        is-allocated-prime-overlay-initiator;

     => relative-addr-of-allocated-name
        (allocated-name-corresponding-to (name-declared-in
        (declaration-for (nm)))) #OTHERWISE #.
```

```
#DF allocated-name-corresponding-to (nm)

     "{nm #IS <name>}"

     => #FIRST alloc-name #IN sequence-of-allocated-names-in
        ( addressing-unit-containing (nm)) #SUCH-THAT (
        #STRING-OF-TERMINALS-OF (alloc-name) #EQW
        #STRING-OF-TERMINALS-OF (nm)) #.


#DF nearest-prior-overlay-reference-to (nm)

     "{ ($ nm $) initiates-its-overlay #AND #NOT ($ nm $)
     is-prime-overlay-initiator }"

     =>overlay-reference-to (nm, "in" preceding
        (overlay-declaration-containing (nm), "in"
        sequence-of-overlay-declarations-mentioning (nm)))
        #.


#DF overlay-reference-to (nm, "in" ovly-dec)

     "{ nm #IS <name> #AND ovly-dec #IS
     <overlay-declaration> }"

     => #FIRST nx #IN (#SEQUENCE-OF <name> #IN ovly-dec)
        #SUCH-THAT (nx #EQW nm) #.


#DF first-name-in-overlay (ovly-dec)

     "{ ($ovly-dec $) is-absolute-overlay-declaration}"

     => #FIRST-ELEMENT-IN ( #SEQUENCE-OF <name> #IN
        ovly-dec) #.


#DF initiates-its-data-sequence (nm)

     "{ nm #IS <name> #AND ($nm$)
     is-an-overlay-declaration}"

     => #TRUE #IFF nm #EQ data-sequence-initiator-of
        (data-sequence-containing (nm)) #.
```

```
#DF overlay-declaration-containing (nm)

     "{ ($nm$) is-in-overlay-declaration}"

     => #LAST ovly-dec #IN (#SEQUENCE-OF-ANCESTORS-OF nm)
        #SUCH-THAT (ovly-dec #IS
        <independent-overlay-declaration> #U
        <subordinate-overlay-declaration>) #.

#DF data-sequence-initiator-rel-addr (nm)

     "{ ($nm$) initiates-its-data-sequence #AND #NOT ($nm$)
     initiates-its-overlay}"

     => alloc-name-addr-for (nm, "at" word-address-from
        (overlay-element-rel-addr (overlay-initiator-of
        (overlay-declaration-containing (nm))))) #.

#DF preceding-this-nm-in-its-data-sequence (nm)

     "{ nm #IS <name> #AND ($nm$)
     is-in-overlay-declaration}"

     => preceding (nm, "in" names-in-data-seq (
        data-sequence-containing (nm))) #.

#DF data-sequence-containing (nm)

     "{ nm #IS <name> #AND ($nm$)
     is-in-overlay-declaration}"

     => #LAST ds #IN (#SEQUENCE-OF-ANCESTORS-OF nm)
        #SUCH-THAT (ds #IS <data-sequence>) #.

#DF names-in-data-seq (ds)

     "{ds #IS <data-sequence> }"

     => #SEQUENCE-OF <name> #IN ds #.

#DF relative-addr-of-allocated-name (alloc-name)

     "{ ($alloc-name$) is-allocated-name #AND #NOT
```

```
    ($alloc-name$) is-prime-overlay-initiator}"

    => alloc-name-addr-for (alloc-name, "at" 0) #IF
       ($alloc-name$) is-first-in-addressing-unit;

    => alloc-name-addr-for (alloc-name, "at" 1 +
       last-word-addr
       (relative-addr-of-structure-designated-by
       (($alloc-name$) preceding-this-allocated-name)))
       #OTHERWISE #.


#DF is-first-in-addressing-unit (alloc-name)

    "{ ($alloc-name$) is-allocated-name}"

    => #TRUE #IFF alloc-name #EQ #FIRST-ELEMENT-IN
       sequence-of-allocated-names-in
       (addressing-unit-containing (alloc-name)) #.


#DF alloc-name-addr-for (alloc-name, "at" n)

    "{ alloc-name #IS <name> #AND n> =0 #AND #NOT
    ($alloc-name$) is-prime-overlay-initiator }"

    => call-by-name-addr-at (n) #IF ($alloc-name$)
       is-that-of-a-call-by-name-parameter;

    =>($basic-addr(alloc-name), "with" n$)
       replacing-first-word-address #OTHERWISE #.


#DF is-that-of-a-call-by-name-parameter (alloc-name)

    "{ ($alloc-name$) is-allocated-name}"

    => #FALSE #IF program-context (alloc-name) #IS-NOT
       <procedure-declaration> #U <procedure-subprogram>;

    => #FALSE #IF program-context (declaration-for
       (alloc-name)) #IS-NOT <procedure-declaration> #U
       <procedure-subprogram>;

    => #TRUE #IFF
       there-is-a-formal-name-matching(alloc-name)
       #OTHERWISE #.
```

```
#DF there-is-a-formal-name-matching (alloc-name)

    "{ ($alloc-name$) is-allocated-name #AND
    program-context (alloc-name) #IS
    <procedure-declaration> #U <procedure-subprogram> }"

    => #THERE-EXISTS formal-nm #IN (#SEQUENCE-OF <name> #IN
       optional-formal-parameter-list-of (procedure-head-of
       (program-context (alloc-name)))) #SUCH-THAT (
       ($formal-nm$) is-not-control-parameter #AND
       formal-nm #EQW alloc-name) #.


#DF is-not-control-parameter (nm)

    "{ nm #IS <name> }"

    => #PARENT-NODE (nm) #IS-NOT
       <formal-input-close-parameter> #U
       <formal-output-destination-parameter> #.


#DF preceding-this-allocated-name (alloc-name)

    "{ ($alloc-name$) is-allocated-name }"

    => preceding (alloc-name, "in"
       sequence-of-allocated-names-in
       (addressing-unit-containing (alloc-name))) #.


#DF call-by-name-addr-at (n)

    "{ n>=0 }"

    => one-word-reference-address-at (n) #.


#DF one-word-reference-address-at (n)

    "{ n>=0 }"

    => create-standard-reference-address
       (build-field-descriptor (n, 0, bits-per-word),
       null-next-field-descriptor, bits-per-word) #.
```

```
#DF replacing-first-word-address (s-ref-addr, "by" n)

     "{ n >= 0 #AND ($s-ref-addr$)
     is-standard-reference-address}"

     => ($s-ref-addr, "with" ($field-descriptor-of
        (s-ref-addr), "having" n$) replacing-word-address $)
        replacing-field-descriptor #.


#DF basic-addr (nm)

     "{ nm #IS <name> }"

     => basic-allocated-addr (nm) #.

#DF basic-allocated-addr (alloc-name)

     "{ ($alloc-name$) is-allocated-name }"

     => basic-mode-addr (detailed-declaration-for
        (alloc-name)) #IF ($alloc-name$) is-mode-declared;

     => basic-simple-item-addr (detailed-declaration-for
        (alloc-name)) #IF ($alloc-name$)
        is-simple-item-reference;

     => basic-array-addr (detailed-declaration-for
        (alloc-name)) #IF ($alloc-name$) is-array-reference;

     => basic-ordinary-table-addr (
        implied-declaration-for(alloc-name)) #IF
        ($alloc-name$) is-ordinary-table-reference;

     => basic-defined-entry-table-addr (
        implied-declaration-for (alloc-name)) #IF
        ($alloc-name$) is-defined-entry-table-reference;

     => basic-like-table-declaration-addr
        (declaration-for(alloc-name)) #IF ($alloc-name$)
        is-like-declared-table #.


#DF is-mode-declared (nx)

     => detailed-declaration-for (nx) #IS <mode-directive>
        #.
```

```
#DF basic-mode-addr (dec)

     "{ dec #IS <mode-directive> }"

     => basic-addr-from-item-descriptor
        (typed-item-description-of (dec)) #.


#DF basic-addr-from-item-descriptor (desc)

     => float-basic-addr #IF desc #IS
        <floating-item-description>;

     => int-basic-addr (integer-specifier-of(desc)) #IF desc
        #IS <integer-item-description>;

     => fix-basic-addr ( fixed-specifier-of(desc)) #IF desc
        #IS <fixed-item-description>;

     => status-basic-addr (bit-count(desc)) #IF desc #IS
        <status-item-description>;

     => boolean-basic-addr #IF desc #IS
        <boolean-item-description> ;

     => literal-basic-addr (byte-count(desc)) #IF desc #IS
        <hollerith-item-description> #U
        <transmission-code-item-description> #.


#DF float-basic-addr

     => create-standard-reference-address
        (build-field-descriptor (0, 0, bits-per-word),
        null-next-field-descriptor, bits-per-word) #.


#DF int-basic-addr (ispec)

     "{ ispec #IS <integer-specifier>}"

     => gen-basic-addr (#STRING-OF-TERMINALS-OF
        (bit-count(ispec))) #.


#DF bit-count (ispec)
```

```
    => #SEG 3 #OF ispec #.

#DF gen-basic-addr (n)
    => create-standard-reference-address
       (build-field-descriptor(0,bits-per-word - n,n),
       null-next-field-descriptor, n) #.

#DF fix-basic-addr (spec)
    "{ spec #IS <fixed-specifier> }"
    => gen-basic-addr (bit-count (spec)) #.

#DF status-basic-addr (n)
    => gen-basic-addr (n) #.

#DF boolean-basic-addr
    => create-standard-reference-address
       (build-field-descriptor (0,35,1),
       null-next-field-descriptor, 1) #.

#DF literal-basic-addr (n)
    => right-justified-literal-basic-addr (n
       *bits-per-byte) #IF n<= bytes-per-word;

    => left-justified-literal-basic-addr (n *
       bits-per-byte) #IF n> bytes-per-word #.

#DF bytes-per-word
    => bits-per-word / bits-per-byte #.

#DF byte-count (desc)
    => #SEG 3 #OF desc #.
```

```
#DF right-justified-literal-basic-addr (n)

     => create-standard-reference-address
        (build-field-descriptor (0, bits-per-word - n,
        bits-per-byte), literal-next-field-descriptor, n) #.


#DF left-justified-literal-basic-addr (n)

     => create-standard-reference-address
        (build-field-descriptor (0, #FIRST-ELEMENT-IN
        byte-boundaries, bits-per-byte),
        literal-next-field-descriptor , n) #.


#DF literal-next-field-descriptor

     => build-next-field-descriptor (bits-per-byte,
        #FIRST-ELEMENT-IN byte-boundaries, 1, bits-per-byte)
        #.


#DF byte-boundaries

     => byte-boundaries-given (bits-per-byte, bits-per-word)
        #.


#DF byte-boundaries-given (n,w)

     "{ n> 0, w>=0 }"

     => #NILSEQ #IF w < n;

     => \w - n\ #CS byte-boundaries-given (n,w - n)
        #OTHERWISE #.


#DF is-simple-item-reference (nx)

     => detailed-declaration-for (nx) #IS
        <simple-item-declaration> #.


#DF basic-simple-item-addr (dec)

     "{ dec #IS <simple-item-declaration> }"
```

```
    => basic-addr-from-item-descriptor
       (typed-item-description-of(dec)) #.


#DF is-array-reference (nx)

    => detailed-declaration-for (nx) #IS
       <array-declaration> #.


#DF basic-array-addr (dec)

    "{ dec #IS <array-dec> }"

    => contiguous-word-ref-addr-of-length
       (number-of-words-in-array(dec) * bits-per-word) #.


#DF contiguous-word-ref-addr-of-length (m "bits")

    "{ m > 0 }"

    => ($basic-contiguous-word-ref-addr, "with" m$)
       replacing-nr-of-bits #.


#DF basic-contiguous-word-ref-addr

    => create-standard-reference-address
       (basic-full-word-descriptor,
       contiguous-word-next-fld-descriptor, #UNDEFINED
       "nr-of-bits") #.


#DF basic-full-word-descriptor

    => build-field-descriptor (0 "word-address", 0
       "field-first-bit", bits-per-word
       "nr-of-bits-in-field") #.


#DF contiguous-word-next-fld-descriptor

    => build-next-field-descriptor (bits-per-word
       "bits-between-field-first-bits", 0
       "first-field-bit", 1 "words-between-fields",
       bits-per-word "nr-of-bits-in-next-field") #.
```

```
#DF number-of-words-in-array (dec)

     "{ dec #IS <array-declaration> }"

     => number-of-columns-in-array(dec) *
        number-of-words-per-column-of-array(dec) #.

#DF number-of-columns-in-array (dec)

     "{ dec #IS <array-declaration>}"

     => dimension-product (all-but-first-element-in
        (dimension-bounds-from (dec))) #.

#DF dimension-bounds-from (dec)

     "{ dec #IS <array-declaration> }"

     => bounds-values (dimension-list-of (dec)) #.

#DF dimension-list-of (dec)

     "{ dec #IS <array-declaration>}"

     => #SEG 7 #OF dec #.

#DF bounds-values (dim-list)

     "{ dim-list #IS <dimension-list>}"

     => \ first-dim-count (dim-list) \ #IF ($dim-list$)
        has-only-one-dimension;

     => \ first-dim-count (dim-list) \ #CS bounds-values
        (rest-of-dimension-list (dim-list)) #OTHERWISE #.

#DF first-dim-count (dim-list)

     "{ dim-list #IS <dimension-list>}"

     => #STRING-OF-TERMINALS-OF (first-dim-of (dim-list)) #.
```

```
#DF has-only-one-dimension (dim-list)

    => #TRUE #IFF dim-list #IS #CASE 1 #OF <dimension-list>
       #.

#DF first-dim-of(dim-list)

    "{ dim-list #IS <dimension-list> }"

    => #SEG 1 #OF dim-list #.

#DF rest-of-dimension-list (dim-list)

    "{ dim-list #IS <dimension-list>}"

    => #SEG 3 #OF dim-list #.

#DF number-of-words-per-column-of-array (dec)

    "{ dec #IS <array-declaration> }"

    => last-word-addr
       (basic-addr-of-boolean-array-column-from (dec)) #IF
       declaration-type (typed-item-description-of (dec))
       #EQW 'boolean';

    => last-word-addr (basic-array-element-addr (dec)) *
       number-of-elements-per-column-of-array (dec)
       #OTHERWISE #.

#DF basic-addr-of-boolean-array-column-from (dec)

    "{ dec #IS <array-declaration> #AND declaration-type
    (typed-item-description-of (dec)) #EQW 'boolean' }"

    => create-standard-reference-address
       (build-field-descriptor (0,0,1),
       build-next-field-descriptor (1,0,1,1),
       number-of-elements-per-column-of-array(dec)) #.

#DF basic-array-element-addr (dec)
```

```
    "{ dec #IS <array-declaration> #AND declaration-type
    (typed-item-description-of (dec)) #NEQW 'boolean'}"

    => basic-addr-from-item-descriptor
       (typed-item-description-of (dec)) #.


#DF number-of-elements-per-column-of-array (dec)

    "{ dec #IS <array-declaration>}"

    => 1 + (#FIRST-ELEMENT-IN dimension-bounds-from (dec))
       #.


#DF implied-declaration-for (nm)

    "{ ($nm$) is-allocated-name}"

    => table-implicitly-referenced-by (nm) #IF ($nm$)
       implicitly-first-refers-to-a-table;

    => declaration-for (nm) #OTHERWISE #.


#DF is-ordinary-table-reference (nm)

    "{ nm #IS <name>}"

    => table-implicitly-referenced-by (nm) #IS
       <ordinary-table-declaration> #IF ($ nm $)
       implicitly-first-refers-to-a-table ;

    => declaration-for (nm) #IS
       <ordinary-table-declaration> #OTHERWISE #.


#DF basic-ordinary-table-addr (dec)

    "{ dec #IS <ordinary-table-declaration>}"

    => contiguous-word-ref-addr-of-length
       (number-of-words-in-ordinary-table(dec) *
       bits-per-word) #.


#DF number-of-words-in-ordinary-table(dec)
```

```
     "{ dec #IS <ordinary-table-declaration>}"

     => one-for-nent-word + number-of-entries-of (dec) *
        number-of-words-per-entry-in (dec) #.


#DF one-for-nent-word

     => 1 #.


#DF is-defined-entry-table-reference (nm)

     "{ nm #IS <name>}"

     => table-implicitly-referenced-by (nm) #IS
        <defined-entry-table-declaration> #IF ($ nm $)
        implicitly-first-refers-to-a-table ;

     => declaration-for (nm) #IS
        <defined-entry-table-declaration> #OTHERWISE #.


#DF basic-defined-entry-table-addr (dec)

     "{ dec #IS <defined-entry-table-declaration>}"

     => contiguous-word-ref-addr-of-length
        (number-of-words-in-defined-entry-table(dec) *
        bits-per-word) #.


#DF number-of-words-in-defined-entry-table(dec)

     "{dec #IS <defined-entry-table-declaration>}"

     => one-for-nent-word + number-of-entries-of (dec) *
        number-of-words-per-entry-in (dec ) #.


#DF is-like-declared-table (nm)

     "{ nm #IS <name>}"

     => table-implicitly-referenced-by (nm) #IS
        <like-table-declaration> #IF ($ nm $)
        implicitly-first-refers-to-a-table ;
```

```
     => #TRUE #IFF declaration-for (nm) #IS
        <like-table-declaration> #AND
        detailed-declaration-for (nm) #IS
        <ordinary-table-declaration> #U
        <defined-entry-table-declaration> #OTHERWISE #.


#DF number-of-entries-of (dec)

     "{ dec #IS <defined-entry-table-declaration> #U
     <ordinary-table-declaration> #U
     <like-table-declaration>}"

     => #CONVERT 10 (#STRING-OF-TERMINALS-OF
        (nominal-size-specifier-of
        (table-size-specification-of (dec)))) #.


#DF nominal-size-specifier-of (tss)

     "{ tss #IS <table-size-specification>}"

     => #SEG 3 #OF tss #.


#DF basic-like-table-declaration-addr (dec)

     "{dec #IS <like-table-declaration>}"

     => contiguous-word-ref-addr-of-length
        (number-of-words-in-like-table(dec) * bits-per-word)
        #.


#DF number-of-words-in-like-table (dec)

     "{ dec #IS <like-table-declaration>}"

     => one-for-nent-word + number-of-entries-of-like-table
        (dec) * number-of-words-per-entry-in (dec) #.


#DF number-of-entries-of-like-table (dec)

     "{ dec #IS <like-table-declaration>}"

     => number-of-entries-of (dec) #IF
```

```
        #STRING-OF-TERMINALS-OF (table-size-specification-of
        (dec)) #NEQW #NIL;

     => number-of-entries-of-like-table
        (pattern-table-decl-for (dec)) #IF
        pattern-table-decl-for(dec) #IS
        <like-table-declaration>;

     => number-of-entries-of (pattern-table-decl-for(dec))
        #OTHERWISE #.


#DF number-of-words-per-entry-in (dec)

     "{ dec #IS <ordinary-table-declaration> #U
     <defined-entry-table-declaration> #U
     <like-table-declaration>}"

     => nr-words-per-entry-in-like-table (dec) #IF dec #IS
        <like-table-declaration>;

     => nr-words-per-entry-in-ordinary-table(dec) #IF dec
        #IS <ordinary-table-declaration>;

     => nr-words-per-entry-in-defined-entry-table (dec) #IF
        dec #IS <defined-entry-table-declaration> #.


#DF nr-words-per-entry-in-like-table (dec)

     "{ dec #IS <like-table-declaration> }"

     => number-of-words-per-entry-in
        (ultimate-pattern-table-decl-for (dec)) #IF
        ultimate-pattern-table-decl-for (dec) #IS
        <defined-entry-table-declaration>;

     => number-of-words-per-entry-in (
        pattern-table-decl-for (dec)) #IF
        optional-packing-specification-of (dec) #EQW #NIL;

     => nr-words-per-dense-entry
        (ultimate-pattern-table-decl-for (dec)) #IF
        optional-packing-specification-of (dec) #EQW 'D';

     => nr-words-per-med-packed-entry
        (ultimate-pattern-table-decl-for (dec)) #IF
        optional-packing-specification-of (dec) #EQW 'M';
```

```
    => nr-words-per-unpacked-entry
       (ultimate-pattern-table-decl-for(dec)) #IF
       optional-packing-specification-of (dec) #EQW 'N' #.


#DF nr-words-per-entry-in-defined-entry-table(dec)

    "{ dec #IS <defined-entry-table-declaration>}"

    => #CONVERT 10 (#STRING-OF-TERMINALS-OF
       (words-per-entry-designator-of (dec))) #.


#DF words-per-entry-designator-of (dec)

    "{ dec #IS <defined-entry-table-declaration>}"

    => #SEG 9 #OF dec #.


#DF nr-words-per-entry-in-ordinary-table(dec)

    "{dec #IS <ordinary-table-declaration>}"

    => nr-words-per-dense-entry(dec) #IF
       #STRING-OF-TERMINALS-OF
       (optional-packing-specification-of(dec)) #EQW 'D';

    => nr-words-per-med-packed-entry(dec) #IF
       #STRING-OF-TERMINALS-OF
       (optional-packing-specification-of(dec)) #EQW 'M';

    => nr-words-per-unpacked-entry (dec) #IF
       #STRING-OF-TERMINALS-OF
       (optional-packing-specification-of (dec)) #IS-IN
       \'N', #NIL\ #.


#DF nr-words-per-unpacked-entry (dec)

    "{ dec #IS <ordinary-table-declaration> }"

    => last-word-addr
       (entry-rel-addr-of-unpacked-structure-designated-by
       (last-allocated-name-in-entry-of (dec) ) ) #.
```

```
#DF last-allocated-name-in-entry-of (dec)

     "{ dec #IS <ordinary-table-declaration> }"

     => #LAST-ELEMENT-IN
        sequence-of-names-allocated-in-entry-of (dec) #.

#DF sequence-of-names-allocated-in-entry-of (dec)

     "{ dec #IS <ordinary-table-declaration> }"

     => #SUBSEQUENCE-OF-ELEMENTS alloc-nm #IN
        sequence-of-allocation-names-in-entry-of (dec)
        #SUCH-THAT ( ($alloc-nm$)
        is-actually-allocated-in-entry) #.

#DF is-actually-allocated-in-entry (alloc-nm)

     "{ ($alloc-nm$) is-allocation-name-in-ordinary-entry}"

     => #TRUE #IF ($alloc-nm$)
        is-prime-subordinate-overlay-initiator;

     => #FALSE #IF ($alloc-nm$)
        is-mentioned-in-a-subordinate-ovly-dec;

     => #TRUE #IF ($alloc-nm$)
        is-declared-subordinate-item-name #.

#DF sequence-of-allocation-names-in-entry-of (dec)

     "{ dec #IS <ordinary-table-declaration> }"

     => #SUBSEQUENCE-OF-ELEMENTS alloc-nm #IN
        sequence-of-possible-allocation-names-in-entry-of
        (dec) #SUCH-THAT ( ($alloc-nm$)
        is-entry-allocation-name) #.

#DF is-entry-allocation-name (alloc-nm)

     "{ alloc-nm #IS <name> }"

     => #TRUE #IFF #PARENT-NODE (alloc-nm) #IS
        <simple-variable> #.
```

```
#DF sequence-of-possible-allocation-names-in-entry-of (dec)

    "{ dec #IS <ordinary-table-declaration> }"

    => #SEQUENCE-OF <name> #IN
       ordinary-entry-description-of (dec) #.


#DF ordinary-entry-description-of (dec)

    "{ dec #IS <ordinary-table-declaration> }"

    => #SEG 13 #OF dec #.


#DF is-prime-subordinate-overlay-initiator (nm)

    "{ ($nm$) is-allocation-name-in-ordinary-entry }"

    => ($nm$) is-prime-subord-initiator #IF ($nm$)
       is-subord-overlay-initiator;

    => #FALSE #OTHERWISE #.


#DF is-subord-overlay-initiator (nm)

    "{ ($nm$) is-allocation-name-in-ordinary-entry }"

    => ($nm, "in" subordinate-ovly-dec-containing (nm) $)
       appears-as-the-subordinate-overlay-initiator #IF
       ($nm$) is-contained-in-a-subordinate-ovly-dec;

    => #FALSE #OTHERWISE #.


#DF is-contained-in-a-subordinate-ovly-dec (nm)

    "{ ($nm$) is-allocation-name-in-ordinary-entry }"

    => #TRUE #IFF #THERE-EXISTS ovly-dec #IN (
       #SEQUENCE-OF-ANCESTORS-OF nm) #SUCH-THAT (nm #IS
       <subordinate-overlay-declaration>) #.


#DF subordinate-ovly-dec-containing (nm)
```

```
    "{ ($nm$) is-contained-in-a-subordinate-ovly-dec }"

    => #LAST ovly-dec #IN ( #SEQUENCE-OF-ANCESTORS-OF nm)
       #SUCH-THAT (nm #IS
       <subordinate-overlay-declaration>) #.

#DF appears-as-the-subordinate-overlay-initiator (nm, "in"
ovly-dec)

    "{ ($nm$) is-allocation-name-in-ordinary-entry #AND
    ovly-dec #IS <subordinate-overlay-dec> #AND nm #IS
    #NODE-IN ovly-dec }"

    => #TRUE #IFF nm #EQN overlay-initiator-of (ovly-dec)
       #.

#DF is-prime-subord-initiator (nm)

    "{ ($nm$) is-subord-overlay-initiator }"

    => nm #EQN overlay-initiator-of ( #FIRST-ELEMENT-IN
       seq-of-subord-overlay-declarations-mentioning (nm) )
       #IF #FOR-ALL ovly-dec #IN
       seq-of-subord-overlay-declarations-mentioning (nm)
       #IT-IS-TRUE-THAT ( ($nm, "in" ovly-dec$)
       matches-the-overlay-initiator) #.

#DF is-mentioned-in-a-subordinate-ovly-dec (nm)

    "{ ($nm$) is-allocation-name-in-ordinary-entry }"

    => #TRUE #IFF
       seq-of-subord-overlay-declarations-mentioning (nm)
       #NEQ #NILSEQ #.

#DF seq-of-subord-overlay-declarations-mentioning (nm)

    "{ ($nm$) is-allocation-name-in-ordinary-entry }"

    => #SUBSEQUENCE-OF-ELEMENTS ovly-dec #IN ( #SEQUENCE-OF
       <subordinate-overlay-declaration> #IN
       ordinary-table-dec-containing (nm) ) #SUCH-THAT (
       ($nm, "in" ovly-dec$) is-mentioned) #.
```

```
#DF ordinary-table-dec-containing (nm)

    => #LAST table-dec #IN ( #SEQUENCE-OF-ANCESTORS-OF nm)
       #SUCH-THAT (table-dec #IS
       <ordinary-table-declaration>) #.

#DF is-declared-subordinate-item-name (nm)

    "{ ($nm$) is-allocation-name-in-ordinary-entry #AND
    #NOT ($nm$) is-mentioned-in-a-subordinate-ovly-dec }"

    => nm #EQN name-declared-in
       (subordinate-item-declaration-containing (nm) ) #.

#DF subordinate-item-declaration-containing (nm)

    => #LAST ancestor #IN ( #SEQUENCE-OF-ANCESTORS-OF nm)
       #SUCH-THAT (ancestor #IS
       <ordinary-table-item-declaration> #U
       <string-item-declaration> #U
       <defined-entry-item-declaration>) #.

#DF entry-rel-addr-of-unpacked-structure-designated-by (nm)

    "{ ($nm$) is-actually-allocated-in-entry }"

    => entry-rel-addr-of-unpacked-ovly-block-initiated-by
       (nm) #IF ($nm$)
       is-prime-subordinate-overlay-initiator;

    => entry-rel-addr-of-unpacked-allocated-name (nm)
       #OTHERWISE #.

#DF entry-rel-addr-of-unpacked-allocated-name (nm)

    "{ ($nm$) is-actually-allocated-in-entry }"

    => alloc-name-addr-for (nm, "at" 0) #IF ($nm$)
       is-first-in-ordinary-entry;

    => alloc-name-addr-for (nm, "at" 1 + last-word-addr
       (entry-rel-addr-of-unpacked-structure-designated-by
```

```
        ( ($nm$)
        preceding-this-allocated-ordinary-entry-name) ) )
        #OTHERWISE #.


#DF is-first-in-ordinary-entry (nm)

     "{ ($nm$) is-actually-allocated-in-entry }"

     => #TRUE #IFF nm #EQ #FIRST-ELEMENT-IN
        sequence-of-names-allocated-in-entry-of
        (ordinary-table-dec-containing (nm) ) #.


#DF preceding-this-allocated-ordinary-entry-name (nm)

     "{ ($nm$) is-actually-allocated-in-entry #AND #NOT
     ($nm$) is-first-in-ordinary-entry }"

     => preceding (nm, "in"
        sequence-of-names-allocated-in-entry-of
        (ordinary-table-dec-containing (nm) ) ) #.


#DF entry-rel-addr-of-unpacked-ovly-block-initiated-by (nm)

     "{ ($nm$) is-actually-allocated-in-entry #AND ($nm$)
     is-prime-subordinate-overlay-initiator }"

     => contiguous-word-ref-addr-at (word-address-from
        (entry-rel-addr-of-unpacked-allocated-name (nm) ).
        "with" length-of-unpacked-overlay-block-initiated-by
        (nm) * bits-per-word "bits") #.


#DF length-of-unpacked-overlay-block-initiated-by (nm)

     "{ ($nm$) is-prime-subordinate-overlay-initiator }"

     => 1 + last-word-addr-in-unpacked-ovly-blk-at (nm) -
        word-address-from
        (entry-rel-addr-of-unpacked-allocated-name (nm) ) #.


#DF last-word-addr-in-unpacked-ovly-blk-at (nm)

     "{ ($nm$) is-prime-subordinate-overlay-initiator }"
```

```
    => maximal-addr-of
       (sequence-of-data-sequence-terminators-from
       (subordinate-ovly-decs-in-block-initiated-by (nm) )
       ) #.


#DF subordinate-ovly-decs-in-block-initiated-by (nm)

    "{ ($nm$) is-prime-subordinate-overlay-initiator }"

    => ovly-decs-in-block-rooted-by
       (subordinate-ovly-dec-containing (nm) ) #.


#DF nr-words-per-med-packed-entry (dec)

    "{ dec #IS <ordinary-table-declaration> }"

    => last-word-addr
       (entry-rel-addr-of-med-packed-structure-designated-by
       (last-allocated-name-in-entry-of (dec) ) ) #.


#DF entry-rel-addr-of-med-packed-structure-designated-by
(nm)

    "{ ($nm$) is-actually-allocated-in-entry }"

    => entry-rel-addr-of-med-packed-ovly-block-initiated-by
       (nm) #IF ($nm$)
       is-prime-subordinate-overlay-initiator:

    => entry-rel-addr-of-med-packed-allocated-name (nm)
       #OTHERWISE #.


#DF entry-rel-addr-of-med-packed-ovly-block-initiated-by
(nm)

    "{ ($nm$) is-actually-allocated-in-entry }"

    => contiguous-bytes-ref-addr-at (first-byte-addr
       (entry-rel-addr-of-med-packed-allocated-name (nm) ),
       "with"
       byte-length-of-med-packed-overlay-block-initiated-by
       (nm) * bits-per-byte "bits") #.
```

```
#DF contiguous-bytes-ref-addr-at (byte-addr, "with" n-bits)

      => ($ ($ ($basic-contiguous-byte-ref-addr, "with"
         word-part (byte-addr) $)
         replacing-first-word-address, "and" bit-part
         (byte-addr) $) replacing-first-bit, "and" n-bits$)
         replacing-nr-of-bits #.


#DF basic-contiguous-byte-ref-addr

      => create-standard-reference-address
         (basic-byte-descriptor,
         contiguous-next-byte-descriptor, #UNDEFINED
         "nr-of-bits") #.


#DF basic-byte-descriptor

      => build-field-descriptor (0 "word-address",
         #FIRST-ELEMENT-IN byte-boundaries "field-first-bit",
         bits-per-byte "nr-of-bits-in-field") #.


#DF contiguous-next-byte-descriptor

      => build-next-field-descriptor (bits-per-byte
         "bits-between-field-first-bits", #FIRST-ELEMENT-IN
         byte-boundaries "first-field-bit", 1
         "words-between-fields", bits-per-byte
         "nr-of-bits-in-next-field") #.


#DF first-byte-addr (s-ref-addr)

      "{ ($s-ref-addr$) is-standard-reference-address }"

      => next-byte-at-or-following (first-bit-addr
         (s-ref-addr) ) #.


#DF byte-length-of-med-packed-overlay-block-initiated-by
(nm)

      "{ ($nm$) is-prime-subordinate-overlay-initiator }"

      => byte-distance-between (first-byte-addr
         (entry-rel-addr-of-med-packed-allocated-name (nm) )
```

```
    , "and" last-byte-addr-in-med-packed-ovly-blk-at
    (nm) ) #.


#DF byte-distance-between (byte-addr1, byte-addr2)

     "{ ($ \ byte-addr1, byte-addr2 \ $) are-byte-addresses
     }"

     => 1+ #ABS (bytes-per-word * (word-part (byte-addr1) -
        word-part (byte-addr2) ) + (nr-bytes-in-wd-before
        (byte-addr1) - nr-bytes-in-wd-before (byte-addr2) )
        ) #.


#DF nr-bytes-in-wd-before (byte-addr)

     "{ ($byte-addr$) is-byte-address }"

     => (#ORDPOSIT (bit-part (byte-addr) ) #IN
        byte-boundaries) - 1 #.


#DF last-byte-addr-in-med-packed-ovly-blk-at (nm)

     "{ ($nm$) is-prime-subordinate-overlay-initiator }"

     => maximal-byte-addr-of-med-packed
        (sequence-of-data-sequence-terminators-from
        (subordinate-ovly-decs-in-block-initiated-by (nm) )
        ) #.


#DF maximal-byte-addr-of-med-packed (ds-term-seq)

     "{ #FOR-ALL x #IN ds-term-seq #IT-IS-TRUE-THAT ( ($x$)
     is-data-seq-terminator }"

     => last-byte-addr (overlay-element-med-packed-rel-addr
        ( #FIRST-ELEMENT-IN ds-term-seq ) ) #IF #LENGTH
        (ds-term-seq) =1;

     => maximum-bit-addr (last-byte-addr
        (overlay-element-med-packed-rel-addr (
        #FIRST-ELEMENT-IN ds-term-seq ) ).
        maximal-byte-addr-of-med-packed
        (all-but-first-element-in (ds-term-seq ) ) )
        #OTHERWISE #.
```

```
#DF overlay-element-med-packed-rel-addr (nm)

    "{nm #IS <name> #AND ($nm$)
    is-in-subordinate-overlay-declaration}"

    => overlay-initiator-med-packed-rel-addr (nm) #IF
       ($nm$) initiates-its-overlay;

    => data-sequence-initiator-med-packed-rel-addr (nm) #IF
       ($nm$) initiates-its-data-sequence;

    => med-packed-allocated-name-addr-for (nm, "at"
       next-byte-after (last-byte-addr
       (overlay-element-med-packed-rel-addr (($nm$)
       preceding-this-nm-in-its-data-sequence))))
       #OTHERWISE #.

#DF overlay-initiator-med-packed-rel-addr (nm)

    "{ ($nm$) initiates-its-overlay #AND ($nm$)
    is-contained-in-a-subordinate-ovly-dec}"

    => entry-rel-addr-of-med-packed-allocated-name (nm) #IF
       ($nm$) is-prime-subordinate-overlay-initiator;

    => overlay-element-med-packed-rel-addr
       (nearest-prior-subord-ovly-reference-to (nm))
       #OTHERWISE #.

#DF data-sequence-initiator-med-packed-rel-addr (nm)

    "{ ($nm$) initiates-its-data-sequence #AND #NOT ($nm$)
    initiates-its-overlay}"

    => med-packed-allocated-name-addr-for (nm, "at"
       first-byte-addr (overlay-element-med-packed-rel-addr
       (overlay-initiator-of
       (overlay-declaration-containing (nm))))) #.

#DF entry-rel-addr-of-med-packed-allocated-name (nm)

    "{ ($nm$) is-actually-allocated-in-entry}"
```

```
     => med-packed-allocated-name-addr-for (nm, "at"
        first-byte-bit-addr) #IF ($nm$)
        is-first-in-ordinary-entry;

     => med-packed-allocated-name-addr-for (nm, "at"
        next-byte-after (last-byte-addr
        (entry-rel-addr-of-med-packed-structure-designated-by
        (($nm$)
        preceding-this-allocated-ordinary-entry-name))))
        #OTHERWISE #.

#DF first-byte-bit-addr

     => bit-addr-built-from (0, #FIRST-ELEMENT-IN
        byte-boundaries) #.

#DF last-byte-addr (s-ref-addr)

     "{ ($s-ref-addr$) is-standard-reference-address}"

     => bit-addr-from-field-descriptor ( #LAST-ELEMENT-IN
        field-descriptor-sequence-implied-in (s-ref-addr))
        #.

#DF bit-addr-from-field-descriptor (fldesc)

     "{ ($fldesc$) is-field-descriptor}"

     => bit-addr-built-from (word-address-of (fldesc),
        field-first-bit-of (fldesc)) #.

#DF next-byte-after (byte-addr)

     "{ ($byte-addr$) is-byte-address}"

     => first-byte-next-word-following (byte-addr) #IF
        bit-part (byte-addr) = #LAST-ELEMENT-IN
        byte-boundaries;

     => next-byte-this-word-following (byte-addr) #OTHERWISE
        #.

#DF med-packed-allocated-name-addr-for (nm, "at" byte-addr)
```

```
    "{ ($byte-addr$) is-byte-address #AND ($nm$)
    is-allocation-name-in-entry}"

    => med-packed-ordinary-table-item-addr
       (detailed-declaration-for (nm) , "at" byte-addr) #.


#DF med-packed-ordinary-table-item-addr (dec,"at"byte-addr)

    "{ ($byte-addr$) is-byte-address #AND dec #IS
    <ordinary-table-item-declaration>}"

    => med-packed-shifted-addr-from-item-descriptor
       (typed-item-description-of (dec), "to" byte-addr) #.


#DF med-packed-shifted-addr-from-item-descriptor
(desc,"to"byte-addr)

    "{ ($desc$) is-typed-item-description #AND
    ($byte-addr$) is-byte-address}"

    => ($ basic-addr-from-item-descriptor (desc) , "to"
       byte-addr$) shifted-ref-addr #IF number-of-bits-in (
       basic-addr-from-item-descriptor (desc) ) + bit-part
       (byte-addr) <= bits-per-word;

    => ($ basic-addr-from-item-descriptor (desc) , "to"
       next-word-first-bit-following (byte-addr) $)
       shifted-ref-addr #OTHERWISE#.


#DF nr-words-per-dense-entry (dec)

    "{ dec #IS <ordinary-table-declaration>}"

    => last-word-addr
       (entry-rel-addr-of-dense-structure-designated-by
       (last-allocated-name-in-entry-of (dec))) #.


#DF entry-rel-addr-of-dense-structure-designated-by (nm)

    "{ ($nm$) is-actually-allocated-in-entry}"

    => entry-rel-addr-of-dense-ovly-block-initiated-by (nm)
       #IF ($nm$) is-prime-subordinate-overlay-initiator:
```

```
     => entry-rel-addr-of-dense-allocated-name (nm)
        #OTHERWISE #.


#DF entry-rel-addr-of-dense-ovly-block-initiated-by (nm)

     "{ ($nm$) is-actually-allocated-in-entry}"

     => contiguous-bits-ref-addr-at (first-bit-addr
        (entry-rel-addr-of-dense-allocated-name (nm)),
        "with"
        bit-length-of-dense-overlay-block-initiated-by (nm)
        "bits") #.


#DF contiguous-bits-ref-addr-at (bit-addr, "with"n"bits")

     "{ ($bit-addr$) is-bit-address #AND n>=0}"

     => ($contiguous-word-ref-addr-at (word-part (bit-addr),
        "with" n "bits" )."with"bit-part(bit-addr)$)
        replacing-first-bit#.


#DF replacing-first-bit (s-ref-addr,"with"n)

     "{ ($s-ref-addr$) is-standard-reference-address #AND
     n>=0}"

     => ($s-ref-addr,"with"
        ($field-descriptor-of(s-ref-addr),"with"n$)
        replacing-field-first-bit$)
        replacing-field-descriptor#.


#DF bit-length-of-dense-overlay-block-initiated-by (nm)

     "{ ($nm$) is-prime-subordinate-overlay-initiator}"

     => bit-distance-between (first-bit-addr
        (entry-rel-addr-of-dense-allocated-name (nm)) ,"and"
        last-bit-addr-in-dense-ovly-blk-at (nm)) #.


#DF first-bit-addr (s-ref-addr)

     "{ ($s-ref-addr$) is-standard-reference-address}"
```

```
     => bit-addr-from-field-descriptor (field-descriptor-of
        (s-ref-addr)) #.


#DF bit-distance-between (bit-addr1, bit-addr2)

     "{ ($ \ bit-addr1, bit-addr2 \ $) are-bit-addresses}"

     => 1+ #ABS (( bit-part (bit-addr1) - bit-part
        (bit-addr2)) + (word-part (bit-addr1) - word-part
        (bit-addr2)) * bits-per-word) #.


#DF last-bit-addr-in-dense-ovly-blk-at (nm)

     "{ ($nm$) is-prime-subordinate-overlay-initiator}"

     => maximal-bit-addr-of-densely-packed
        (sequence-of-data-sequence-terminators-from
        (subordinate-ovly-decs-in-block-initiated-by (nm)))
        #.


#DF maximal-bit-addr-of-densely-packed (ds-term-seq)

     "{ #FOR-ALL x #IN ds-term-seq #IT-IS-TRUE-THAT (($x$)
     is-data-seq-terminator}"

     => last-bit-addr (overlay-element-dense-rel-addr (
        #FIRST-ELEMENT-IN ds-term-seq)) #IF #LENGTH
        (ds-term-seq) =1;

     => maximum-bit-addr (last-bit-addr
        (overlay-element-dense-rel-addr ( #FIRST-ELEMENT-IN
        ds-term-seq)), maximal-bit-addr-of-densely-packed
        (all-but-first-element-in (ds-term-seq))) #OTHERWISE
        #.


#DF maximum-bit-addr (bit-addr1, bit-addr2)

     "{ ($) bit-addr1, bit-addr2\$) are-bit-addresses}"

     => bit-addr1 #IF word-part (bit-addr1) > word-part
        (bit-addr2);

     => bit-addr1 #IF bit-part(bit-addr1) > bit-part
```

```
     (bit-addr2);
     => bit-addr2 #OTHERWISE #.

#DF overlay-element-dense-rel-addr (nm)

    "{ nm #IS<name> #AND ($nm$) is-in-overlay-declaration}"

    => overlay-initiator-dense-rel-addr (nm) #IF ($nm$)
       initiates-its-overlay;

    => data-sequence-initiator-dense-rel-addr (nm) #IF
       ($nm$) initiates-its-data-sequence;

    => dense-allocated-name-addr-for (nm, "at"
       next-bit-after (last-bit-addr
       (overlay-element-dense-rel-addr ( ($nm$)
       preceding-this-nm-in-its-data-sequence))))
       #OTHERWISE #.

#DF overlay-initiator-dense-rel-addr (nm)

    "{ ($nm$) initiates-its-overlay #AND ($nm$)
    is-contained-in-a-subordinate-ovly-dec}"

    => entry-rel-addr-of-dense-allocated-name (nm) #IF
       ($nm$) is-prime-subordinate-overlay-initiator;

    => overlay-element-dense-rel-addr (
       nearest-prior-subord-ovly-reference-to (nm))
       #OTHERWISE #.

#DF data-sequence-initiator-dense-rel-addr (nm)

    "{ ($nm$) initiates-its-data-sequence #AND #NOT ($nm$)
    initiates-its-overlay}"

    => dense-allocated-name-addr-for (nm, "at"
       first-bit-addr (overlay-element-dense-rel-addr
       (overlay-initiator-of
       (overlay-declaration-containing (nm))))) #.

#DF entry-rel-addr-of-dense-allocated-name (nm)
```

```
    "{ ($nm$) is-actually-allocated-in-entry}"

    => dense-allocated-name-addr-for (nm,
       "at"zero-bit-addr) #IF ($nm$)
       is-first-in-ordinary-entry;

    => dense-allocated-name-addr-for (nm, "at"
       next-bit-after (last-bit-addr
       (entry-rel-addr-of-dense-structure-designated-by
       (($nm$)
       preceding-this-allocated-ordinary-entry-name))))
       #OTHERWISE #.

#DF zero-bit-addr

    => bit-addr-built-from (0,0) #.

#DF last-bit-addr (s-ref-addr)

    "{ ($s-ref-addr$) is-standard-reference-address}"

    => last-bit-addr-from-field-descriptor (
       #LAST-ELEMENT-IN
       field-descriptor-sequence-implied-in (s-ref-addr))
       #.

#DF last-bit-addr-from-field-descriptor (fld-desc)

    "{ ($fld-desc$) is-field-descriptor}"

    => bit-addr-built-from (word-address-of (fld-desc),
       (field-first-bit-of (fld-desc) +
       nr-of-bits-in-field-of (fld-desc) - 1)) #.

#DF next-bit-after (bitaddr)

    "{ ($bitaddr$) is-bit-address}"

    => next-word-first-bit-following (bitaddr) #IF bit-part
       (bitaddr) >= bits-per-word - 1;

    => bit-addr-built-from (word-part (bitaddr), bit-part
       (bitaddr) +1) #OTHERWISE #.
```

```
#DF next-word-first-bit-following (bit-addr)

     => bit-addr-built-from (word-part (bit-addr)+1,0) #.


#DF dense-allocated-name-addr-for (nm, "at" bitaddr)

     "{ ($bitaddr$) is-bit-address #AND ($nm$)
     is-allocation-name-in-entry}"

     => dense-ordinary-table-item-addr
        (detailed-declaration-for (nm) , "at" bitaddr) #.


#DF dense-ordinary-table-item-addr (dec,"at"bitaddr)

     "{ ($bitaddr$) is-bit-address #AND dec #IS
     <ordinary-table-item-declaration>}"

     => dense-shifted-addr-from-item-descriptor
        (typed-item-description-of (dec) , "to"bitaddr) #.


#DF dense-shifted-addr-from-item-descriptor (desc, bitaddr)

     "{ ($desc$) is-typed-item-description #AND ($bitaddr$)
     is-bit-address}"

     => dense-char-shift (basic-addr-from-item-descriptor
        (desc), "to" bitaddr) #IF desc #IS
        <hollerith-item-description> #U
        <transmission-code-item-description>;

     => dense-non-char-shift
        (basic-addr-from-item-descriptor (desc) , "to"
        bitaddr) #OTHERWISE #.


#DF dense-char-shift (s-ref-addr, "to"bit-addr)

     => ($s-ref-addr, "to"next-byte-at-or-following
        (bit-addr)$) shifted-ref-addr #.


#DF next-byte-at-or-following (bit-addr)

     "{ ($bit-addr$) is-bit-address)" => bit-addr #IF
```

```
     bit-part (bit-addr) #IS-IN byte-boundaries;  =>
     first-byte-next-word-following (bit-addr) #IF bit-part
     (bit-addr) > #LAST-ELEMENT-IN byte-boundaries;  =>
     next-byte-this-word-following (bit-addr) #OTHERWISE #.
     #DF first-byte-next-word-following (bit-addr) "{
     ($bit-addr$) is-bit-address}"

     => bit-addr-built-from (word-part (bit-addr),
        #FIRST-ELEMENT-IN byte-boundaries) #.


#DF next-byte-this-word-following (bit-addr)

     "{ ($bit-addr$) is-bit-address #AND bit-part (bit-addr)
     < #LAST-ELEMENT-IN byte-boundaries}"

     => bit-addr-built-from (word-part (bit-addr), #FIRST
        boundary #IN byte-boundaries #SUCH-THAT
        (boundary>bit-part (bit-addr))) #.


#DF shifted-ref-addr (s-ref-addr, "to"bit-addr)

     => ($s-ref-addr, "with" shifted-fld-descriptor
        (field-descriptor-of (s-ref-addr) , "to" bit-addr)
        $) replacing-field-descriptor #.


#DF shifted-fld-descriptor (fld-desc, "to"bit-addr)

     => ($($fld-desc, "with" word-part (bit-addr) $)
        replacing-word-address, "and" bit-part (bit-addr) $)
        replacing-field-first-bit #.


#DF dense-non-char-shift (s-ref-addr, bitaddr)

     => ($ s-ref-addr, "to" bitaddr$) shifted-ref-addr #IF
        number-of-bits-in (s-ref-addr) + bit-part (bitaddr)
        <= bits-per-word;

     => ($ s-ref-addr, "to" next-word-first-bit-following
        (bitaddr)$) shifted-ref-addr #OTHERWISE #.
```

==

MISSION

of

Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

AMERICAN REVOLUTION BICENTENNIAL 1776-1976